IBM Z OMEGAMON Data Provider 1.1

Installation and User's Guide

Note:

Before using this information and the product it supports, read “Product legal notices” on page 215.

2024-07-11 edition

This edition applies to IBM Z® OMEGAMON® Data Provider Version 1.1 with the PTF for APAR OA66572, and to all subsequent releases and modifications until otherwise indicated in new editions.

Figures

Tables

Contents

About this document

This document describes how to install, configure, and use OMEGAMON Data Provider.

What's new in OMEGAMON Data Provider

A summary of significant changes.

July 2024: APAR OA66572

2024 Q2 maintenance.

OMEGAMON Data Broker

No changes to OMEGAMON Data Broker.

OMEGAMON Data Connect

HTTP output: batching

The new `batching` configuration parameter enables sending multiple records per HTTP POST request. Batching offers higher throughput than the previous single-record-per-request behavior.

For example:

```
http:
  enabled: true
  endpoints:
    ep1:
      url: http://example.com:9997
      batching:
        enabled: true # Default: false (single record per request, as
before)
        # The following parameters apply only when batching is enabled
        batch-size: 5000 # Default: 1000
        linger: 1s # Default: 250ms
```

If batching is enabled, then the output sends up to `batch-size` records per request, according to the following rules:

- If the number of records in the queue is at least `batch-size`, then the output sends a request containing `batch-size` records.
- If there are fewer than `batch-size` records in the queue, then the output waits for up to the duration specified by `linger`. If the number of records in the queue reaches `batch-size` within that duration, then the output sends a request containing `batch-size` records. Otherwise, if there are still fewer than `batch-size` records in the queue at the end of that duration, then the output sends a request containing however many records are in the queue.

Previously, the HTTP output always sent only a single record per request, which remains the default behavior.

HTTP output: compression

The new `compression` configuration parameter enables gzip compression of HTTP POST request bodies.

Compression is particularly useful when batching is enabled.

Example configuration:

```
http:
  enabled: true
  endpoints:
    ep1:
      url: http://example.com:9997
      compression: true # Default: false
```

```
      batching:
        enabled: true
```

HTTP output: custom request headers

The new `headers` configuration parameter enables you to add custom headers to HTTP POST requests.

For example, the following configuration:

```
http:
  enabled: true
  endpoints:
    ep1:
      url: http://example.com:9997
      headers: # Custom headers to add to each HTTP POST request
        - key: odp_header_key_1
          value: header_value_1
        - key: odp_header_key_2
          value: header_value_2
```

adds the following headers to each request:

```
odp_header_key_1: header_value_1
odp_header_key_2: header_value_2
```

Some analytics platforms use custom headers to determine how to process a request. For details, see the documentation for your analytics platform.

Prometheus output: custom metric expiration

The new `metric-expiration` configuration parameter enables you to control when OMEGAMON Data Connect expires (removes) metrics from the Prometheus endpoint.

`metric-expiration` applies per table:

```
connect:
  output:
    prometheus:
      enabled: true
      mappings:
        products:
          <product_code>:
            enabled: true
            tables:
              <table_name>:
                enabled: true
                metric-expiration: <seconds>
                metrics: ...
```

The `metric-expiration` parameter is optional. For data from OMEGAMON monitoring agents, the default metric expiration behavior is unchanged.

May 2024: APAR OA66223

2024 Q1 maintenance.

Important: If you are an existing user of OMEGAMON Data Provider, this APAR introduces changes that require you to perform upgrade steps.

OMEGAMON Data Broker

New version of forwarder protocol

The OMEGAMON Data Broker forwarder protocol is a bespoke TCP/IP-based protocol for forwarding data to OMEGAMON Data Connect.

This APAR introduces version 2 of the forwarder protocol. The original protocol is now known as version 1. By default, OMEGAMON Data Broker now uses version 2 of the protocol.

Important: Upgrade all of your instances of OMEGAMON Data Broker and OMEGAMON Data Connect at the same time. Earlier APAR levels of OMEGAMON Data Connect cannot receive data that is sent using protocol version 2. For example, if you upgrade OMEGAMON Data Broker but not OMEGAMON Data Connect, then OMEGAMON Data Connect will no longer receive data.

Version 2 includes the following enhancements:

Improved performance
OMEGAMON Data Broker now sends a batch of records in each message to OMEGAMON Data Connect.

Previously, using version 1, OMEGAMON Data Broker sent only a single record in each message.

Avoiding data loss under certain conditions
OMEGAMON Data Broker no longer discards unsent records in the forwarder queue if the connection to OMEGAMON Data Connect is lost.

Previously, using version 1, if OMEGAMON Data Broker lost connection to OMEGAMON Data Connect, then OMEGAMON Data Broker discarded unsent records in the forwarder queue.

Now, if OMEGAMON Data Connect reconnects before the forwarder queue fills, then OMEGAMON Data Broker does not lose data.

Hence, depending on the data rate and the time it takes to restart OMEGAMON Data Connect, you can now restart OMEGAMON Data Connect without causing data loss in OMEGAMON Data Broker.

Version 2 introduces two new optional OMEGAMON Data Broker configuration parameters: `PROTOCOL_VERSION` and `MAX_BATCH_SIZE`. Typically, you don't need to be aware of these parameters.

Store parameters for queue capacity and cells are now optional
The following parameters are now optional:

`KAY.CIDB.STORE.<store_id>.QUEUE.CAPACITY`
`KAY.CIDB.STORE.<store_id>.CELL.<cell_id>.SIZE`
`KAY.CIDB.STORE.<store_id>.CELL.<cell_id>.CAPACITY`

These parameters expose implementation details that you typically don't need to know. Unless you have site-specific reasons to specify non-default values, you should remove these parameters from your configuration members.

Reusable sink parameters
You can now define *reusable* sink parameters separately from forwarders. Multiple forwarders can refer to the same set of reusable sink parameters. Reusable sink parameters make it easier to forward data from different sources to the same sink (the same instance of OMEGAMON Data Connect). For example:

```
* Stores
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON
KAY.CIDB.STORE.STORE2.NAME=SOURCEB

* New: reusable sink parameters
KAY.CIDB.SINK.SINK1.HOST=analytics1.example.com
KAY.CIDB.SINK.SINK1.PORT=15351
KAY.CIDB.SINK.SINK1.SECURITY=TLSv1.2
KAY.CIDB.SINK.SINK1.FIPS=ON
KAY.CIDB.SINK.SINK1.KEYRING=KAYSIS01/KAYring

* Forwarders that refer to reusable sink parameters
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1
```

```
KAY.CIDB.FWD.FWD2.SOURCE_STORE=SOURCEB
KAY.CIDB.FWD.FWD2.SINK=SINK1
```

Previously, to define multiple forwarders that use the same sink, you had to duplicate the sink parameters for each forwarder. For example (for conciseness, this example omits store queue capacity and cell parameters, which are now optional):

```
* Stores
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON
KAY.CIDB.STORE.STORE2.NAME=SOURCEB

* Forwarders (with duplicate sink parameters)
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK_HOST=analytics1.example.com
KAY.CIDB.FWD.FWD1.SINK_PORT=15351
KAY.CIDB.FWD.FWD1.SECURITY=TLSv1.2
KAY.CIDB.FWD.FWD1.FIPS=ON
KAY.CIDB.FWD.FWD1.KEYRING=KAYSIS01/KAYring

KAY.CIDB.FWD.FWD2.SOURCE_STORE=SOURCEB
KAY.CIDB.FWD.FWD2.SINK_HOST=analytics1.example.com
KAY.CIDB.FWD.FWD2.SINK_PORT=15351
KAY.CIDB.FWD.FWD2.SECURITY=TLSv1.2
KAY.CIDB.FWD.FWD2.FIPS=ON
KAY.CIDB.FWD.FWD2.KEYRING=KAYSIS01/KAYring
```

Sink parameters that are specified with a forwarder are still supported, and are now known as *forwarder-scope* sink parameters.

Improved validation of configuration parameters

OMEGAMON Data Broker now validates more configuration parameters at startup. Warning message KAYB0023W now reports more bad parameter values.

OMEGAMON Data Connect

Support for new version of OMEGAMON Data Broker forwarder protocol

OMEGAMON Data Connect supports the new version 2 of the OMEGAMON Data Broker forwarder protocol.

Instances of OMEGAMON Data Connect that have been upgraded to this APAR level can receive data using either protocol version 1 or 2. That is, upgraded instances of OMEGAMON Data Connect can receive data from earlier APAR levels of OMEGAMON Data Broker.

For more details, see the corresponding change for OMEGAMON Data Broker.

Improved validation of configuration parameters

Previously, OMEGAMON Data Connect silently ignored unknown parameters in the configuration file. That behavior included ignoring parameters that were known in some contexts, but unknown in the context they were specified in the file. The configuration file is a YAML document, so the indentation of a parameter determines its context. Incorrect indenting could cause issues that were difficult to troubleshoot.

For example, suppose you specify the following parameters with the intention of enabling output to STDOUT:

```
connect:
  output:
    stdout:
    enabled: true # Incorrectly indented
```

These parameters won't have the intended effect, because the `enabled` key is unknown at the same indent level as `stdout`. Instead, `enabled` should be indented as a child of `stdout`. Previously,

OMEGAMON Data Connect silently ignored `enabled: true` in this context. The result was that output to STDOUT was not enabled.

Now, if the configuration file contains an unknown parameter, OMEGAMON Data Connect fails at startup.

Attention: If your configuration file contains errors that were previously silently ignored, this is a breaking change. You must edit the configuration file and fix the issue.

Configuration validator

The OMEGAMON Data Provider configuration validator is a new command-line tool that you can use to validate the two YAML-format OMEGAMON Data Provider configuration files:

- The OMEGAMON Data Connect configuration file
- If you are using OMEGAMON monitoring agents as a data source: the OMEGAMON Data Provider collection configuration member, RKANPARU(KAYOPEN)

After editing these configuration files, you can run the configuration validator as a "preflight" check before using the files.

Sample shell script z/OS® UNIX file now encoded using EBCDIC code page 1047

You no longer need to activate automatic text conversion to run the sample shell script to start OMEGAMON Data Connect on z/OS UNIX.

Previously, the script was supplied on z/OS UNIX tagged as a text file that is uniformly encoded in code set ISO8859-1. While this encoding and tagging is a common practice for text files on z/OS UNIX, by default z/OS UNIX interprets shell scripts using EBCDIC encoding. To run the script on z/OS UNIX, you had to activate automatic text conversion of tagged UNIX file system files by setting the _BPXK_AUTOCVT environment variable to ON. Otherwise, when you ran the script, z/OS UNIX attempted to interpret the script using EBCDIC encoding and reported the error message FSUM7332 `syntax error`.

Documentation-only changes

Configuring OMEGAMON Data Broker

OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. You can configure OMEGAMON Data Broker using one of the following servers:

- The server that is supplied with OMEGAMON Data Provider
- An existing server in a separate Zowe installation

This documentation now recommends the server that is supplied with OMEGAMON Data Provider as the best and easiest choice for most users, with reasons.

Previously, a single procedure covered both options, with conditional steps for each option. Now, each option has its own simpler procedure.

December 2023: APAR OA65724

Upgraded Zowe cross-memory server, raised minimum version of Java™.

Important: If you are an existing user of OMEGAMON Data Provider, this APAR introduces changes that require you to perform upgrade steps.

Breaking changes

Breaking changes introduced by APAR OA65724.

OMEGAMON Data Broker

Minimum supported Zowe version raised from 1.28.2 to 2.12.0

OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. The latest plug-in requires a server from Zowe 2.12.0 or later.

This is a breaking change only if both of the following conditions are true:

- You are running OMEGAMON Data Broker using a Zowe cross-memory server that you have obtained from a separate Zowe distribution.
- The version of that Zowe distribution is earlier than 2.12.0.

In that case, you need to use a Zowe cross-memory server load module from a more recent version of Zowe.

OMEGAMON Data Provider supplies the Zowe cross-memory server load module from Zowe 2.12.0. For details, see the description of the change "Upgraded Zowe cross-memory server".

New required configuration parameter to register the ZISDYNAMIC plug-in

The latest OMEGAMON Data Broker plug-in requires ZISDYNAMIC, the ZIS dynamic linkage base plug-in.

OMEGAMON Data Provider supplies the ZISDYNAMIC plug-in as `TKANMODP(KAYSISDL)`.

You need to register the ZISDYNAMIC plug-in by adding a new parameter to your Zowe cross-memory server configuration member. For example, in `PARMLIB(KAYSIP`*xx*`)`:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
```

The supplied sample member `TKANSAM(KAYSIP00)` has been updated to include this new parameter.

OMEGAMON Data Connect

Minimum supported Java version raised from 8 to 17

OMEGAMON Data Connect is a Java application developed using the Spring Boot framework.

Previously, OMEGAMON Data Connect used a version of Spring Boot that is based on Java 8. That version of Spring Boot has an end of support date of 24 November 2023.

To continue to take advantage of ongoing Spring Boot support, such as bug fixes, security updates, and new features, OMEGAMON Data Connect now uses a more recent version of Spring Boot that is based on Java 17.

You must upgrade the Java runtime environment that you use for OMEGAMON Data Connect to Java 17 or later, 64-bit edition.

Running OMEGAMON Data Connect on z/OS now requires z/OS 2.5 or later. Java 17 runtime environments on z/OS require IBM® Semeru Runtime Certified Edition for z/OS 17. IBM Semeru Runtime Certified Edition for z/OS 17 requires z/OS 2.5 or later. Java 17 is not available on z/OS 2.4 or earlier.

Important: If you are currently running OMEGAMON Data Connect on a version of z/OS earlier than 2.5, then do not apply the PTF for this APAR until you have decided where and how to run OMEGAMON Data Connect afterward. For example:

- Run OMEGAMON Data Connect on a z/OS 2.5 or later LPAR. OMEGAMON Data Connect can run on a different LPAR or sysplex than OMEGAMON Data Broker. For details, see “OMEGAMON Data Provider topology” on page 39.
- Run OMEGAMON Data Connect on a non-z/OS platform that supports Java 17.
- Defer applying the PTF for this APAR of OMEGAMON Data Provider until you upgrade to z/OS 2.5 or later. The z/OS 2.4 end of support (EOS) date is 30 September 2024.

In Spring Boot server SSL properties, follow `safkeyring:` with two (2) slashes, not four (4)

If you run OMEGAMON Data Connect on z/OS, you can use the `safkeyring` protocol to refer to a RACF® key ring.

You must now follow `safkeyring:` with only two (2) slashes.

Previously, in Spring Boot server SSL properties (`server.ssl.*`), you had to follow `safkeyring:` with four (4) slashes.

Kafka producer property names must be enclosed in square brackets and double quotes

To ensure that OMEGAMON Data Connect correctly parses Kafka producer properties in the OMEGAMON Data Connect configuration file, you must enclose the property names in square brackets, and then in double quotes. Example:

```
connect:
  output:
    kafka:
      properties:
        "[reconnect.backoff.max.ms]": 30000
```

Kafka output parameters `retry-interval` and `max-connection-attempts` removed

Previously, to configure the reconnection behavior of the Kafka output, you could specify optional `retry-interval` and `max-connection-attempts` parameters:

```
connect:
  output:
    kafka:
      retry-interval: <seconds> # Default: 30
      max-connection-attempts: <number> # Default: unlimited
```

`retry-interval` and `max-connection-attempts` are no longer supported. They have been superseded by the enhanced internal queueing introduced by this APAR.

Instead, to configure the reconnection behavior of the Kafka output, specify native Kafka producer configuration properties such as `reconnect.backoff.max.ms`:

```
connect:
  output:
    kafka:
      properties:
        "[reconnect.backoff.max.ms]": 30000
```

For information about Kafka producer configuration properties, see the Apache Kafka documentation.

Other changes

Other (non-breaking) changes introduced by APAR OA65724.

OMEGAMON Data Broker

Upgraded Zowe cross-memory server

OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. OMEGAMON Data Provider supplies the Zowe cross-memory server load module in TKANMODP(KAYSIS01). The load module supplied with OMEGAMON Data Provider has been updated to match the load module supplied with Zowe 2.12.0.

New z/OS MVS™ `MODIFY` system command to dynamically change logging level of OMEGAMON Data Broker network activity

Typically, you only need to set this logging level if IBM Software Support requests you to do so for troubleshooting.

Previously, you could only set this logging level in the configuration member of the Zowe cross-memory server.

Now, you can enter a **`MODIFY`** system command to change this logging level dynamically, while OMEGAMON Data Broker is running.

OMEGAMON Data Connect

Kafka output now supports RACF key rings as a store type for SSL

If you run OMEGAMON Data Connect on z/OS, you can now use RACF key rings as keystores or truststores to configure a secure (SSL/TLS) connection to Kafka.

Enhanced internal queueing

Each output, including each TCP sink and each HTTP endpoint, now has its own internal queue.

Previously, all outputs sharing a single internal queue.

Under normal circumstances, where OMEGAMON Data Connect processes data as fast as it arrives, and all destinations read data as fast as it arrives from OMEGAMON Data Connect, there is no significant change in behavior.

This enhancement introduces a change in behavior only when the number of queued events (records) reaches the queue capacity.

Previously, a problem with one output could cause all outputs to lose data. If the single internal queue reached capacity, OMEGAMON Data Connect stopped reading incoming data, exerting back pressure on OMEGAMON Data Broker.

OMEGAMON Data Broker responds to back pressure with the following behavior:

1. OMEGAMON Data Broker stops sending data to OMEGAMON Data Connect.
2. OMEGAMON Data Broker continues to accept incoming data, causing its forwarder queues to fill up.

   When a queue is full, each new incoming record overwrites the oldest remaining record in the queue. Records overwritten in OMEGAMON Data Broker are lost to *all* outputs of OMEGAMON Data Connect.

For example, previously, if an output destination was connected to OMEGAMON Data Connect but stopped reading incoming data because it was out of space, then the single internal queue in OMEGAMON Data Connect could fill to capacity. The subsequent behavior of OMEGAMON Data Connect and OMEGAMON Data Broker could potentially cause *all* outputs to lose data.

Now, a problem with one OMEGAMON Data Connect output does not affect other outputs. If the queue for an output is at capacity, the queue rejects (drops) any new incoming records. Regardless of whether any queues are at capacity, OMEGAMON Data Connect continues to accept incoming data and does not exert back pressure.

Reduced default queue capacity

The configuration parameter `connect.event-publisher.queue-capacity` specifies the maximum number of records in a queue. The default value has been reduced from 1,000,000 (one million) to 50,000 (fifty thousand).

The previous much higher default value, allowing up to one million queued events, could potentially mask data throughput issues and delay troubleshooting. The reduced value helps to identify data throughput issues earlier.

New message KAYC0084W warns you when a queue reaches capacity

If an internal queue for an output reaches capacity, the queue rejects (drops) any new incoming records. The output loses data, hence the warning.

Increased Java heap size

In the sample JCL procedure and shell script to run OMEGAMON Data Connect, the Java runtime options for minimum and maximum heap size have been increased from:

```
-Xms64m -Xmx2048m
```

to:

```
-Xms1024m -Xmx4096m
```

This increase is a precautionary measure to accommodate higher data volume, including larger incoming record sizes.

Actual heap size requirements depend on factors that are specific to your site.

Documentation-only changes

Acknowledging data sources other than OMEGAMON monitoring agents

Previously, the only data sources for OMEGAMON Data Provider were OMEGAMON monitoring agents.

Now, OMEGAMON Data Provider also has other data sources:

- IBM Db2® Automation Expert for z/OS
- IBM Db2 Query Monitor for z/OS

Some introductory topics have been updated to acknowledge these other data sources and distinguish between content that applies to all data sources and content that applies only to OMEGAMON monitoring agents. However, later topics do not yet acknowledge these other data sources.

September 2023: APAR OA65247

OMEGAMON Data Connect has been enhanced with a new HTTP output for Instana.

New HTTP output

The new HTTP output sends POST requests to HTTP or HTTPS endpoints such as IBM Instana Observability on z/OS (Instana). Each request contains a JSON object that describes a single record (event).

The JSON from the HTTP output has the same structure as the existing TCP and Kafka outputs. The HTTP output supports the same security (SSL/TLS) and filtering parameters as those existing outputs.

July 2023: APAR OA64880

Architectural enhancement to enable alternative deployment options for attributes support.

Important: If you are an existing user of OMEGAMON Data Provider, this APAR introduces changes that require you to perform upgrade steps.

Overview of changes:

OMEGAMON Data Broker

- Updated Zowe cross-memory server load module, from Zowe 1.28.2, removes the previous restriction on the load module name.

OMEGAMON Data Connect

- The previous monolithic JAR file has been split into one *mapping extension* JAR file per agent and one *core* JAR file.

  This new modular architecture enables alternative deployment options for attributes support, in addition to the conventional option of supplying attributes support with OMEGAMON Data Provider.
- New *user directories* keep your site-specific configuration details separate from the installation directory.

  To run OMEGAMON Data Connect, you now refer to a user directory. A user directory contains a configuration file and, optionally, mapping extension JAR files.
- Logging flood control suppresses duplicate messages.
- Improved runtime exception handling offers more control and consistency for SpEL expressions in filters.
- Refreshed attribute support introduces the latest attributes from agents.

For details, see the descriptions of breaking changes and other changes.

Documentation-only changes:

"Prerequisites for OMEGAMON Data Provider"
Lists new ways to get OMEGAMON Data Provider.

"Where and how to run OMEGAMON Data Connect"
A new topic that describes how you can run OMEGAMON Data Connect, a Java application, on various platforms. Includes subtopics that describe the installation and user directories, and how to use the supplied sample JCL procedure and shell script to run OMEGAMON Data Connect.

Breaking changes

Breaking changes introduced by APAR OA64880.

OMEGAMON Data Broker

Minimum supported Zowe version raised from 1.24 to 1.28.2
OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. The latest plug-in requires a server from Zowe 1.28.2 or later.

This is a breaking change only if both of the following conditions are true:

- You are running OMEGAMON Data Broker using a Zowe cross-memory server that you have obtained from a separate Zowe distribution.
- The version of that Zowe distribution is earlier than 1.28.2.

In that case, you need to use a Zowe cross-memory server load module from a more recent version of Zowe.

OMEGAMON Data Provider supplies the Zowe cross-memory server load module from Zowe 1.28.2. For details, see the description of the change "Updated Zowe cross-memory server".

OMEGAMON Data Connect

Renamed JAR file to odp-server-*version*.jar
The file `data-connect-`*`version`*`.jar` has been renamed to `odp-server-`*`version`*`.jar`. This file is now known as the *core* JAR file, to distinguish it from *mapping extension* JAR files.

New symbolic link odp-server.jar
The `lib` directory that contains the core JAR file now also contains a symbolic link, `odp-server.jar`, that refers to the core JAR file. The core JAR file name contains a version. The symbolic link offers a stable file name that avoids the inconvenience of updating references to the core JAR file whenever the version changes.

New runtime option -Dodp.ext refers to new mapping extension JAR files
Previously, a monolithic JAR file implemented the core functions of OMEGAMON Data Connect *and* the support for each monitoring agent. Now, there are multiple JAR files:

lib/odp-server-*version*.jar
The core JAR file: a single JAR file that implements the core functions of OMEGAMON Data Connect.

lib/ext/k*pp*-odp-model-*version*.jar
Mapping extension JAR files: a JAR file for each monitoring agent supported by OMEGAMON Data Provider, where k*pp* is the product code for the agent. Each mapping extension JAR file encapsulates the support for a specific agent.

Mapping extensions extend OMEGAMON Data Connect to support different types of incoming data. Mapping extensions consist of Java classes that contain the data and logic required to map binary-format data from monitoring agents to various output formats. These classes are sometimes referred to as *mapping classes*.

The new OMEGAMON Data Connect runtime option `-Dodp.ext` specifies the locations of mapping extensions as a comma-separated list of directory paths and individual JAR file paths. The sample JCL procedure and shell script for running OMEGAMON Data Connect set a default value

for `-Dodp.ext` that includes the `lib/ext` directory under the installation directory and the `extensions` directory under the user directory. If a mapping extension JAR file for an agent exists in more than one location, then OMEGAMON Data Connect uses the latest version of the JAR file.

Changes to the sample shell script

Changes to the sample shell script, `bin/connect`, include the following breaking changes:

New required argument to specify an action

Previously, you could run the script with no command-line argument. Now, you must specify one of the following actions as a command-line argument:

`run`
Runs OMEGAMON Data Connect, as before.

`create`
Creates a user directory for OMEGAMON Data Connect containing a sample configuration file copied from the installation directory as a starting point for you to edit.

This is the recommended method for creating a user directory, even if you plan to use JCL to run OMEGAMON Data Connect.

New environment variable `ODP_CONNECT_USER_DIR`

The script refers to a user directory specified by the environment variable `ODP_CONNECT_USER_DIR`. Set this variable before running the script.

Other changes

Other (non-breaking) changes introduced by APAR OA64880.

OMEGAMON Data Broker

Updated Zowe cross-memory server

OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. OMEGAMON Data Provider supplies the Zowe cross-memory server load module in `TKANMODP(KAYSIS01)`. The load module supplied with OMEGAMON Data Provider has been updated to match the load module supplied with Zowe 1.28.2. This new version introduces various changes:

No need to rename the supplied load module from `KAYSIS01` to `ZWESIS01`

Previously, you had to rename the supplied module to `ZWESIS01` before using it. The module relied on that name to load itself into the link pack area (LPA).

Now, the module no longer relies on that name. You no longer have to rename the supplied module. You can use the supplied member name `KAYSIS01` or a name of your choice. A benefit of this change is that you can run the Zowe cross-memory server directly from the `TKANMODP` SMP/E target library. In the JCL that runs the Zowe cross-memory server, the **`STEPLIB`** concatenation can simply refer to the single `TKANMODP` library, which contains all of the required load modules: the server load module and the load modules for the OMEGAMON Data Broker plug-in.

`PARMLIB` member name matches the first four characters of the load module name

OMEGAMON Data Broker configuration parameters are stored in the configuration member of the Zowe cross-memory server in a `PARMLIB` data set.

Previously, the `PARMLIB` member name was `ZWESIP`*xx*, where *xx* is the value of the optional **`MEM`** runtime parameter in the startup JCL for the Zowe cross-memory server. The default value of **`MEM`** is `00`. Hence, the default `PARMLIB` member name was `ZWESIP00`.

Now, the `PARMLIB` member name is *pppp*`IP`*xx*, where *pppp* matches the first four characters of the Zowe cross-memory server load module name. If you use the load module name `KAYSIS01` as supplied with OMEGAMON Data Provider and the default **`MEM`** value of `00`, then the `PARMLIB` member name is `KAYSIP00`.

Line continuations in PARMLIB member: support moved from plug-in to server

Support for line continuations in the PARMLIB member has moved from the OMEGAMON Data Broker plug-in to the Zowe cross-memory server. There is no change to the supported line continuation syntax. If you use the latest OMEGAMON Data Broker plug-in with the latest Zowe cross-memory server load module supplied with OMEGAMON Data Provider, you will not notice any difference in behavior.

Sample JCL procedure renamed to KAYSIS01 and updated

The sample OMEGAMON Data Broker startup JCL procedure TKANSAM(ZWESIS01) has been renamed to KAYSIS01. The new name reflects the fact that the corresponding supplied load module TKANMODP(KAYSIS01) no longer needs to be renamed to ZWESIS01 before use.

Previously, the sample JCL procedure ZWESIS01 was a verbatim copy of the procedure supplied with Zowe . That procedure referred to the load module ZWESIS01 and specified the **NAME** parameter value ZWESIS_STD.

Now, the renamed and updated sample KAYSIS01 refers to the load module KAYSIS01 and specifies the **NAME** parameter value ODP_BROKER.

The new **NAME** parameter value ODP_BROKER reflects the fact that you might choose to run more than one instance of the Zowe cross-memory server on the same instance of z/OS. For example:

- A server dedicated to running OMEGAMON Data Broker
- Another server running as part of a separate Zowe installation, unrelated to OMEGAMON Data Provider

Each Zowe cross-memory server on the same instance of z/OS must specify a unique **NAME** parameter value. The broker.name key of the OMEGAMON Data Provider collection configuration member, RKANPARU(KAYOPEN), must refer to the instance of the Zowe cross-memory server that runs OMEGAMON Data Broker.

Sample configuration member renamed to KAYSIP00

The sample OMEGAMON Data Broker configuration member TKANSAM(KAYBRP00) has been renamed to KAYSIP00. The new name matches the PARMLIB member name that is required if you use the Zowe cross-memory server load module supplied with OMEGAMON Data Provider with its original name, KAYSIS01, and you use the default **MEM** runtime parameter value, 00.

Value of broker.name key in sample collection configuration member changed to ODP_BROKER

Previously, the broker.name key in the sample OMEGAMON Data Provider collection configuration member TKANSAM(KAYOPEN) specified the value ZWESIS_STD, matching the **NAME** parameter in the sample OMEGAMON Data Broker startup JCL procedure.

Now, broker.name specifies the value ODP_BROKER, matching the **NAME** parameter in the updated sample JCL procedure.

OMEGAMON Data Connect

Changes to the sample JCL procedure and shell script

There are numerous changes to the sample OMEGAMON Data Connect startup JCL procedure sample/KAYCONN (now also supplied in the MVS library TKANSAM) and shell script bin/connect. Compare the new versions of these supplied files to your edited copies, and update your copies accordingly.

Common changes to both samples:

New user directories separate site-specific configuration files from the installation directory

Previously, the sample JCL procedure and shell script assumed that your site-specific configuration file, connect.yaml, was stored in a config subdirectory under the OMEGAMON Data Connect *installation* directory.

The problem: that assumption risked your site-specific changes to connect.yaml being overwritten when you applied service to the installation directory. You had to be careful to not overwrite your edited connect.yaml with the latest sample file.

Now, the sample JCL procedure and shell script make it easier to keep your site-specific configuration files separate from the installation directory.

In the JCL procedure, the symbol KAYHOME, which referred to the OMEGAMON Data Connect installation directory, has been renamed to INSTLDIR. A new USERDIR symbol refers to the user directory containing your configuration file.

Similarly, in the shell script, the new environment variable ODP_CONNECT_USER_DIR refers to the user directory.

Each instance of OMEGAMON Data Connect now refers to a user directory. User directories can be shared.

-jar option refers to the new symbolic link

Rather than referring to the original JAR file name, which contains a version, the -jar option in the Java command line that runs OMEGAMON Data Connect now refers to the new symbolic link, with the stable file name odp-server.jar. Referring to the symbolic link avoids the inconvenience of updating the -jar option when the version changes in the original JAR file name.

Filter condition expressions: improved handling of runtime exceptions

To conditionally filter records, you can write expressions in the Spring Expression Language (SpEL). These expressions can trigger runtime exceptions. For example, if an expression uses an integer field as the denominator in a division operation, then a zero value for that field in an incoming record will trigger a divide-by-zero runtime exception.

Previously, OMEGAMON Data Connect handled runtime exceptions that SpEL characterized as *evaluation exceptions* differently than other runtime exceptions:

1. For all runtime exceptions: discard the current record that triggered the exception.
2. Depending on the type of runtime exception:

   Evaluation exceptions

   a. *Stop* processing records that use the expression; disable the table for outputs that use this filter.

   b. Report messages KAYC0048E and KAYC0056I.

   Other runtime exceptions

   a. *Continue* processing records that use the expression.

   b. Report message KAYC0031W.

In that previous behavior, an expression that triggered an evaluation exception was considered to be unreliable, and was immediately excluded from processing to avoid the potential for undesirable filtering. However, in practice, that behavior is problematic, inconsistent, because other types of runtime exceptions could also be considered to mark the expression as unreliable. For details on which runtime exceptions SpEL characterizes as an evaluation exception, see the SpEL documentation.

Now, OMEGAMON Data Connect handles all runtime exceptions in expressions in the same way. A new configuration parameter for each condition, disable-table-on-error, enables you to control whether OMEGAMON Data Connect stops or continues processing records that use the expression:

1. Discard the current record that triggered the exception.
2. Report new message KAYC0057W.
3. Depending on the value of disable-table-on-error:

   false (default)

   Continue processing records that use the expression.

`true`

a. *Stop* processing records that use the expression; disable the table for outputs that use this filter.

b. Report message KAYC0056I.

Message KAYC0048E no longer occurs.

Example use of the new configuration parameter in a global-level filter:

```
connect:
  filter:
    enabled: true
    products:
      km5:
        tables:
          ascpuutil:
            condition:
              expression: sysplex_name.equals('PLEXA') # No safe
navigation operator (?) after sysplex_name
              disable-table-on-error: true # If sysplex_name field is
missing, stop processing records from this table
```

Logging flood control to suppress duplicate messages

Some events can occur frequently, resulting in numerous duplicate log messages. To avoid duplicate messages flooding the log, OMEGAMON Data Connect applies new flood control configuration parameters to some messages. Here are the new parameters shown with their default values:

```
connect:
  logging:
    flood-control:
      enabled: true
      interval: 300
      limit: 1
```

where `interval` specifies a number of seconds (300 seconds = 5 minutes) and `limit` is the maximum instances of a particular message allowed within that interval.

OMEGAMON Data Connect applies flood control to the following messages:

KAYC0031W
KAYC0057W
KAYC0061W
KAYC0062W

Actuator endpoints over HTTP: now exposing only the `health` endpoint by default

Previously, OMEGAMON Data Connect specified the following Spring Boot property value to expose all Spring Boot Actuator endpoints over HTTP by default:

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

OMEGAMON Data Connect no longer specifies that property. OMEGAMON Data Connect now follows the default Spring Boot behavior, exposing only the `health` endpoint over HTTP.

To control which Spring Boot Actuator endpoints are exposed, use the corresponding `include` or `exclude` properties in the OMEGAMON Data Connect configuration file. For example, to expose the `health` and `prometheus` endpoints over HTTP:

```
management:
  endpoints:
    web:
      exposure:
        include: "health,prometheus"
```

For more details, see the Spring Boot documentation about exposing endpoints for production-ready features.

Removal of z/OS MVS `MODIFY` system command to restart OMEGAMON Data Connect
Previously, if you ran OMEGAMON Data Connect as a z/OS batch job or started task, then you could enter the following MVS **`MODIFY`** system command to restart OMEGAMON Data Connect without stopping the batch job or started task:

```
F job_name,APPL=RESTART
```

That application-specific restart method, using the parameter APPL=RESTART, is no longer supported. Instead, use the normal z/OS methods for stopping and starting batch jobs and started tasks, such as the MVS **STOP** system command and, for started tasks, the **START** system command. For details, see "Reloading OMEGAMON Data Connect configuration" on page 84.

Refreshed attribute support
Attributes support refreshed to include new attributes introduced by monitoring agents.

February 2023: APAR OA64177

Attributes support refreshed to include new attributes introduced by monitoring agents.

No documentation updates.

September 2022: APAR OA63539

Attributes support refreshed to include new attributes introduced by monitoring agents.

Documentation-only changes:

"OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider" on page 42
A new topic about choosing the destinations of collected attributes.

"Expected messages" on page 184
A new topic that lists the normal messages that you should expect from each component involved in OMEGAMON Data Provider.

"Configuration parameters for OMEGAMON monitoring agents as a data source" on page 89

- More details about the special `interval` value 0
- Clarification of default destinations for unselected collections
- Precedence of entries in the `collections` sequence
- More examples

"Adding more OMEGAMON collections to OMEGAMON Data Provider" on page 87
A new topic about adding more collections to an environment that already sends some collections to OMEGAMON Data Provider.

OMEGAMON Data Broker

Forwarding to multiple instances of OMEGAMON Data Connect
Each instance of OMEGAMON Data Broker can forward attributes to multiple instances of OMEGAMON Data Connect.

For an overview of this concept, see “OMEGAMON Data Provider topology” on page 39.

For configuration details, see “OMEGAMON Data Broker configuration parameters” on page 95.

Logging level
Typically, you only need to set the OMEGAMON Data Broker logging options parameter (LOGOPTS) if IBM Software Support requests you to do so for troubleshooting.

OMEGAMON Data Connect

Handling of errors in filter condition expressions
Clarification of how OMEGAMON Data Connect handles different types of errors in filter condition expressions.

Parameters for managing attempts to connect to a TCP sink
Descriptions of two previously undocumented TCP output parameters: `max-connection-attempts` and `retry-interval`.

Methods for setting the logging level
Different ways to set the OMEGAMON Data Connect logging level.

June 2022: APAR OA63141

Support for MQ, Network, and Storage monitoring agents, and other new function.

Support for more monitoring agents

- MQ
  - IBM OMEGAMON for Messaging on z/OS, V7.5
- Network
  - IBM Z OMEGAMON Network Monitor, V5.6
- Storage
  - IBM OMEGAMON for Storage on z/OS, V5.5

Support for Instana
IBM Instana Observability on z/OS can now ingest attributes from OMEGAMON Data Provider as JSON Lines over TCP.

To support this new Instana feature, OMEGAMON Data Connect now includes an embedded filter include file tailored for Instana.

OMEGAMON Data Connect
New configuration parameters:

Filter include files
Filters for JSON-format outputs (Kafka, STDOUT, and TCP) can use the new `include` parameter to refer to an external *filter include file*, rather than specifying filter parameters inline in the OMEGAMON Data Connect configuration file.

The filter include file can be in the file system or embedded in the OMEGAMON Data Connect JAR file.

Conditional filters
Filters for JSON-format outputs can now include a condition for each table.

A condition specifies an expression written in the Spring Expression Language (SpEL). The expression can refer to fields in the table, enabling you to conditionally filter records based on their field values. OMEGAMON Data Connect forwards a record only if the expression is true.

For example, the following parameters configure OMEGAMON Data Connect to send records from the z/OS monitoring agent (product code km5) table `ascpuutil` to the `stdout` file only if the value of the `job_name` field matches the regular expression PFX.*:

```
connect:
  output:
    stdout:
      enabled: true
      filter:
        products:
          km5:
            tables:
              ascpuutil:
                condition:
                  expression: job_name?.matches('PFX.*')
```

Kafka topic per table

Previously, to configure OMEGAMON Data Connect to send data to Kafka, you used the `connect.output.kafka.topic` key to specify the name of a single destination Kafka topic.

Now, the `connect.output.kafka.topic` key is optional:

- If you specify the `topic` key, then the behavior is unchanged: OMEGAMON Data Connect sends data from all tables to that single topic.
- If you omit the `topic` key, then OMEGAMON Data Connect sends data for each table to a separate topic.

  The per-table topic names have the following pattern:

  `topic_prefix.product.table_name`

  where *topic_prefix* is the value of the new key `connect.output.kafka.topic-prefix` (default: odp).

  Example topic name:

  `odp.km5.ascpuutil`

Kafka connection retries after timeout

The following new parameters control retries after an attempt to connect to Kafka times out:

`retry-interval`
Number of seconds between retries.

max-connection-attempts
Maximum number of connection attempts.

Attribute dictionary

OMEGAMON Data Connect now includes an attribute dictionary. The dictionary is a set of YAML files that describe the attributes of each table of each supported monitoring agent.

Documentation-only changes:

- OMNIMON Base APAR/PTF level cited as a prerequisite for OMEGAMON Data Provider.
- Improved description of OMEGAMON Data Provider as a Prometheus target.

March 2022: APAR OA62775

Support for IMS and JVM monitoring agents, and other new function.

Support for more monitoring agents

- IMS:
  - IBM OMEGAMON for IMS on z/OS, V5.5
- Java Virtual Machine (JVM):

- IBM Z OMEGAMON for JVM on z/OS, V5.5

OMEGAMON Data Broker

New warning messages report records lost due to the OMEGAMON Data Broker record queue limit being reached: KAYB0046W, KAYB0047W.

OMEGAMON Data Connect

Multiple TCP outputs

Previously, OMEGAMON Data Connect could send JSON Lines over TCP to only a *single* destination. To send to multiple TCP outputs, you had to run multiple instances of OMEGAMON Data Connect.

Now, a single instance of OMEGAMON Data Connect can send JSON Lines over TCP to *multiple* destinations.

Different filter for each output

Previously, you could specify only a *global-level* filter that applies to all JSON-format outputs: TCP, Kafka, and STDOUT.

Now, you can also specify a filter for each output. These are known as *output-level* filters. If you specify a filter at both levels, the output-level filter replaces the global-level filter.

The combination of multiple TCP outputs and output-level filters means, for example, that a single instance of OMEGAMON Data Connect can send one set of attributes over TCP to Splunk and a different set to the Elastic Stack.

Starter dashboards

The starter Elastic Kibana dashboards have moved to a new GitHub repository.

Documentation-only changes:

- Character encoding issues for the YAML documents `RKANPARU(KAYOPEN)` and `connect.yaml`.
- Updated example Elastic Stack configuration:
  - Uses data streams instead of time-based indices
  - Index names now also include the product code as a qualifier, in addition to the existing table name qualifier

December 2021: APAR OA62420

Support for CICS® and Db2 monitoring agents, and other new function.

Support for more monitoring agents

- CICS:
  - IBM OMEGAMON for CICS on z/OS, 5.5
  - IBM OMEGAMON for CICS TG on z/OS, 5.5
- Db2:
  - IBM OMEGAMON for Db2 Performance Expert on z/OS, 5.4

OMEGAMON Data Connect

Enhanced validation of field and table names in configuration parameters.

OMEGAMON Data Broker

- Configuration member now supports parameters longer than 80 characters
- New configuration parameters for retrying the connection with OMEGAMON Data Connect:

  ```
  KAY.CIDB.FWD.OM.CONNECT_RETRY_INTERVAL
  KAY.CIDB.FWD.OM.MAX_CONNECT_RETRY_ATTEMPTS
  ```

- Support for IPv6 addresses
- Writes significant messages to the JES log

November 2021: First release

OMEGAMON Data Provider was introduced as a part of IBM Z OMEGAMON Integration Monitor.

Earlier editions of this documentation

Earlier editions of this documentation are available in PDF only.

Related information

PDF documentation

Upgrading to the latest APAR level

Some APARs involve performing upgrade steps in addition to the steps described in the corresponding APAR text.

May 2024: APAR OA66223

To upgrade an existing OMEGAMON Data Provider environment to use the PTF for this APAR, you must perform upgrade steps after applying the PTF.

Before you begin

Read what's new in this APAR so that you understand the reasons for these upgrade steps.

Apply the PTF to your z/OS SMP/E target libraries.

Upgrading OMEGAMON Data Connect

Upgrade *all* instances of OMEGAMON Data Connect to this APAR level now, to ensure that all instances can receive data from the upgraded OMEGAMON Data Broker. Use the new configuration validator command-line tool to validate your OMEGAMON Data Connect configuration files.

About this task

Procedure

1. Ensure that the updated OMEGAMON Data Connect installation directory supplied with the PTF for this APAR is used by *all* instances of OMEGAMON Data Connect.

   For example, if you run OMEGAMON Data Connect on platforms other than z/OS, then ensure that you use the files in the updated SMP/E target location of OMEGAMON Data Connect on z/OS to refresh the OMEGAMON Data Connect installation directories on those other platforms.

   For details on the SMP/E target location of OMEGAMON Data Connect, see “Installing OMEGAMON Data Provider” on page 49.
2. Use the new configuration validator to validate your OMEGAMON Data Connect configuration files.

   This APAR introduces improved validation of configuration parameters. Previously, OMEGAMON Data Connect ignored unknown parameters in the configuration file. Now, if the configuration file contains an unknown parameter, OMEGAMON Data Connect fails at startup.

   This APAR also introduces the configuration validator, a command-line tool that you can use to validate the OMEGAMON Data Connect configuration file before starting OMEGAMON Data Connect.

   Use the configuration validator now to reveal errors in the configuration file that OMEGAMON Data Connect was previously silently ignoring, and also later, whenever you edit the configuration file, as a "preflight" check before starting OMEGAMON Data Connect.

   For example, at a shell prompt, change to the `bin` directory under the OMEGAMON Data Connect installation directory, and then enter the following command:

   ```
   ./validate -t=connect /var/omdp/prod-a/config/connect.yaml
   ```

   replacing `/var/omdp/prod-a/config/connect.yaml` with your configuration file path.
3. If the configuration validator reports errors, edit the configuration file to fix the errors, and then run the configuration validator again. Repeat until the configuration validator reports that the configuration file is valid.

Related reference

Configuration validator
The OMEGAMON Data Provider configuration validator is a command-line tool that you can use to validate the OMEGAMON Data Provider configuration files that are in YAML format: the OMEGAMON Data Connect configuration file and, if you are using OMEGAMON monitoring agents as a data source, the OMEGAMON Data Provider collection configuration member, RKANPARU(KAYOPEN).

Related information

OMEGAMON Data Connect fails due to unknown configuration parameters

Upgrading OMEGAMON Data Broker

Remove the now-optional OMEGAMON Data Broker store parameters from the Zowe cross-memory server configuration member.

Procedure

Edit your OMEGAMON Data Broker configuration members and remove the now-optional parameters.

Unless you have site-specific reasons to specify non-default values, you should remove these parameters from your configuration members:

```
KAY.CIDB.STORE.<store_id>.QUEUE.CAPACITY
KAY.CIDB.STORE.<store_id>.CELL.<cell_id>.SIZE
KAY.CIDB.STORE.<store_id>.CELL.<cell_id>.CAPACITY
```

Removing these parameters means that any future changes to their default values will automatically apply when you restart OMEGAMON Data Broker.

Restarting OMEGAMON Data Provider

After applying the PTF and performing the upgrade steps, restart the components involved, and then check for expected messages. In particular, check the messages that report the APAR number.

Procedure

1. Stop any running instances of OMEGAMON Data Broker.
2. Stop any running instances of OMEGAMON Data Connect.
3. Start OMEGAMON Data Connect.

   Tip: If OMEGAMON Data Connect fails to start, it might be because of the improved configuration parameter validation. See “OMEGAMON Data Connect fails due to unknown configuration parameters” on page 180.
4. Start OMEGAMON Data Broker.
5. If you are using OMEGAMON agents as a data source, then restart your OMEGAMON runtime environments.
6. Check for expected messages.

   In particular, check the following messages for the correct APAR number, OA66223:

   OMEGAMON Data Broker
   In the SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker (for example, job name KAYS* or ZWES*):

Message	Check
KAYB0005I	APAR number

   OMEGAMON Data Connect
   In the STDOUT file of OMEGAMON Data Connect:

Message	Check
KAYC0035I	APAR number

December 2023: APAR OA65724

To upgrade an existing OMEGAMON Data Provider environment to use the PTF for this APAR, you must perform upgrade steps after applying the PTF.

Before you begin

Read what's new in this APAR so that you understand the reasons for these upgrade steps.

Running OMEGAMON Data Connect on z/OS now requires z/OS 2.5 or later. Java 17 runtime environments on z/OS require IBM Semeru Runtime Certified Edition for z/OS 17. IBM Semeru Runtime Certified Edition for z/OS 17 requires z/OS 2.5 or later. Java 17 is not available on z/OS 2.4 or earlier.

Important: If you are currently running OMEGAMON Data Connect on a version of z/OS earlier than 2.5, then do not apply the PTF for this APAR until you have decided where and how to run OMEGAMON Data Connect afterward. For example:

- Run OMEGAMON Data Connect on a z/OS 2.5 or later LPAR. OMEGAMON Data Connect can run on a different LPAR or sysplex than OMEGAMON Data Broker. For details, see “OMEGAMON Data Provider topology” on page 39.
- Run OMEGAMON Data Connect on a non-z/OS platform that supports Java 17.
- Defer applying the PTF for this APAR of OMEGAMON Data Provider until you upgrade to z/OS 2.5 or later. The z/OS 2.4 end of support (EOS) date is 30 September 2024.

If, after understanding what's new in this APAR, you decide to proceed, then apply the PTF to your z/OS SMP/E target libraries.

Upgrading OMEGAMON Data Connect

Upgrade the Java runtime environment that you use to run OMEGAMON Data Connect to Java 17 or later, 64-bit edition. Consider increasing the heap size of the Java virtual machine that you use to run OMEGAMON Data Connect. If you use OMEGAMON Data Connect to send data to Apache Kafka, revise the Kafka output configuration parameters.

Before you begin

Ensure that you have a Java 17 or later, 64-bit edition runtime environment installed on the platform where you run OMEGAMON Data Connect. For example, if you run OMEGAMON Data Connect on z/OS, ensure that you have IBM Semeru Runtime Certified Edition for z/OS 17 installed. IBM Semeru Runtime Certified Edition for z/OS 17 requires z/OS 2.5 or later.

Locate the path of the Java installation directory. For example, on z/OS UNIX , `/usr/lpp/java/J17.0_64/`.

Procedure

1. If you use a z/OS JCL procedure to run OMEGAMON Data Connect, upgrade the procedure to refer to a Java 17 runtime environment.
   a) Replace your procedure with the latest sample procedure supplied in `TKANSAM(KAYCONN)`.

      Significant changes in the latest sample procedure:

      `VERSION`
      In the `PROC` statement at the start of the procedure, the value of the `VERSION` parameter is now 17. The procedure uses this value as the suffix of the Java virtual machine (JVM) load module name, `JVMLDM`*xx*.

LIBPATH
In the PARMS inline data set, the value of the LIBPATH environment variable has been updated to match the directories listed in the JZOS documentation for Java 17.

-Djava.protocol.handler.pkgs
The value of the Java runtime option `java.protocol.handler.pkgs` has been updated to `com.ibm.crypto.zsecurity.provider`. This change is described later, in step "3" on page 26.

b) Edit the values of the symbolic parameters at the start of the procedure.

JAVAHOME
The path of the installation directory of Java 17 or later, 64-bit edition. The directory must contain a `bin` subdirectory that contains the **java** command.

INSTLDIR
The path of the OMEGAMON Data Connect installation directory.

USERDIR
The path of your OMEGAMON Data Connect user directory.

c) Ensure that the system can locate the JVMLDM17 load module.

The sample procedure does not contain a **STEPLIB** DD statement to specify the location of the load module. Instead, the procedure assumes that the load module is in a library that the system searches for programs, such as a library in the LNKLST concatenation. For example, a default SMP/E installation of JZOS places the load module in the SYS1.SIEALNKE library, which is in the default LNKLST concatenation.

If the load module is not in such a system library, then insert a **STEPLIB** DD statement that specifies the location of the load module. For example, *hlq*.JZOS.LOADLIB.

2. If you use a shell script or Java command line to run OMEGAMON Data Connect, upgrade them to use a Java 17 or later, 64-bit edition runtime environment.

   This step does not necessarily involve editing the shell script or command line. For example, the supplied sample shell script refers to the JAVA_HOME environment variable; in the profile that sets that variable, change the value to refer to a Java 17 runtime environment.

3. If you run OMEGAMON Data Connect on z/OS, and you specify the Java runtime option `java.protocol.handler.pkgs`, then set the option value to `com.ibm.crypto.zsecurity.provider`.

   ```
   -Djava.protocol.handler.pkgs=com.ibm.crypto.zsecurity.provider
   ```

   (note the `zsecurity` qualifier in the value)

   The Java runtime option `java.protocol.handler.pkgs` refers to packages that contain protocol handlers. For example, a handler for the `safkeyring` protocol that you can use in OMEGAMON Data Connect SSL parameters on z/OS to refer to RACF key rings.

   Previously, in Java 8, the `safkeyring` protocol handler was supplied by the IBMJCE provider in the package `com.ibm.crypto.provider`.

   In Java 17, the IBMJCE provider is no longer supported. The `safkeyring` protocol handler is now supplied by the IBMZSecurity provider in the package `com.ibm.crypto.zsecurity.provider`.

   Tip: Java only reads the value of the `java.protocol.handler.pkgs` runtime option if it encounters a reference to an "unknown" protocol, such as `safkeyring`. Otherwise, Java ignores the value of this runtime option.

4. In each of the methods that you use to run OMEGAMON Data Connect, consider increasing the Java heap size to match the latest samples provided with OMEGAMON Data Connect.

In the sample JCL procedure and shell script to run OMEGAMON Data Connect, the Java runtime options for minimum and maximum heap size have been increased from:

```
-Xms64m -Xmx2048m
```

to:

```
-Xms1024m -Xmx4096m
```

This increase is a precautionary measure to accommodate higher data volume, including larger incoming record sizes.

Actual heap size requirements depend on factors that are specific to your site.

5. If you run OMEGAMON Data Connect on z/OS, and your OMEGAMON Data Connect configuration contains Spring Boot server SSL properties that refer to the `safkeyring` protocol, then reduce the number of slashes following `safkeyring:` from four (4) to two (2).

   For example, given the following `key-store` property value:

   ```
   server:
     ssl:
       key-store: safkeyring:////STCOMDP/OMDPring
   ```

   remove two of the slashes:

   ```
       key-store: safkeyring://STCOMDP/OMDPring
   ```

6. If you use OMEGAMON Data Connect to send data to Apache Kafka, revise the Kafka output configuration parameters.

   a) If you specify Kafka producer properties (that is, parameters under the key `connect.output.kafka.properties`), then wrap the property names in square brackets, and then in quotes.

   To ensure that OMEGAMON Data Connect correctly parses Kafka producer properties in the OMEGAMON Data Connect configuration file, you must enclose the property names in square brackets, and then in double quotes. Example:

   ```
   connect:
     output:
       kafka:
         properties:
           "[reconnect.backoff.max.ms]": 30000
   ```

   b) If you specify the optional Kafka output parameters `retry-interval` and `max-connection-attempts`, then remove those parameters.

   Previously, to configure the reconnection behavior of the Kafka output, you could specify optional `retry-interval` and `max-connection-attempts` parameters:

   ```
   connect:
     output:
       kafka:
         retry-interval: <seconds> # Default: 30
         max-connection-attempts: <number> # Default: unlimited
   ```

   `retry-interval` and `max-connection-attempts` are no longer supported. They have been superseded by the enhanced internal queueing introduced by this APAR.

Instead, to configure the reconnection behavior of the Kafka output, specify native Kafka producer configuration properties such as `reconnect.backoff.max.ms`:

```
connect:
  output:
    kafka:
      properties:
        "[reconnect.backoff.max.ms]": 30000
```

For information about Kafka producer configuration properties, see the Apache Kafka documentation.

Related concepts

Where and how to run OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Related reference

Kafka output parameters
OMEGAMON Data Connect Kafka output parameters specify whether to publish data in JSON format to an Apache Kafka topic.

Related information

OMEGAMON Data Connect fails with UnsupportedClassVersionError

Upgrading OMEGAMON Data Broker

Upgrade the Zowe cross-memory server that runs OMEGAMON Data Broker, and register the additional ZISDYNAMIC plug-in that OMEGAMON Data Broker now requires.

About this task

These upgrade steps are required because OMEGAMON Data Broker now requires:

- A Zowe cross-memory server load module from Zowe 2.12.0 or later.
- The ZIS dynamic linkage base plug-in, ZISDYNAMIC.

If you already run OMEGAMON Data Broker in the Zowe cross-memory server that is supplied with OMEGAMON Data Provider, then you can simply continue doing that. OMEGAMON Data Provider supplies a newer server load module in the same location as the old load module, `TKANMODP(KAYSIS01)`, and the new ZISDYNAMIC plug-in in the same library, as `TKANMODP(KAYSISDL)`.

However, if you run OMEGAMON Data Broker in a Zowe cross-memory server that is part of a separate installation of Zowe earlier than 2.12.0, then either:

- Upgrade your separate installation of Zowe to a more recent version. For details, see the Zowe documentation.

  Ensure that the **STEPLIB** concatenation for your Zowe cross-memory server job step includes the latest OMEGAMON Data Broker load modules, supplied in `TKANMODP(KAYB0001)` and `TKANMODP(KAYBNETL)`.
- Switch to using the server load module that is supplied with OMEGAMON Data Provider. For details on configuring OMEGAMON Data Broker to run in the Zowe cross-memory server that is supplied with OMEGAMON Data Provider, see “Configuring OMEGAMON Data Broker” on page 54.

Whether you use the server supplied with OMEGAMON Data Provider or in a stand-alone installation of Zowe, perform the following procedure to register the ZISDYNAMIC plug-in.

Procedure

1. Register the ZISDYNAMIC plug-in in the Zowe cross-memory server configuration member.

- If you are using the server supplied with OMEGAMON Data Provider, insert the following line at the top of the PARMLIB(KAYSIP*xx*) configuration member:

  ```
  ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
  ```

- If you are using the server in a separate installation of Zowe, insert the following line at the top of the PARMLIB(ZWESIP*xx*) configuration member:

  ```
  ZWES.PLUGIN.ZISDYNAMIC=ZWESISDL
  ```

  Important: Only register the ZISDYNAMIC plug-in once. For example, do not attempt to register the ZISDYNAMIC plug-in once referring to the ZWESISDL load module supplied with Zowe, and then again referring to the KAYSISDL load module supplied with OMEGAMON Data Provider. You do not need to copy the TKANMODP(KAYSISDL) member supplied with OMEGAMON Data Provider to the **STEPLIB** concatenation for the server in your separate installation of Zowe. If you use a server in a separate installation of Zowe, refer to the ZWESISDL plug-in load module supplied with Zowe.

2. Change the name of the existing parameter that registers the OMEGAMON Data Broker plug-in in the Zowe cross-memory server configuration member.

   This change is not strictly necessary, but it's recommended for consistency with the name of the new parameter for the ZISDYNAMIC plug-in.

   Change the existing line:

   ```
   ZWES.PLUGIN.CIDB=KAYB0001
   ```

   to:

   ```
   ZWES.PLUGIN.KAY.CIDB=KAYB0001
   ```

   That is, insert the new qualifier .KAY between .PLUGIN and .CIDB.

Related tasks

Configuring OMEGAMON Data Broker
OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. You can configure OMEGAMON Data Broker using either the server that is supplied with OMEGAMON Data Provider (the best and easiest choice for most users) or a server in an existing separate Zowe installation.

Restarting OMEGAMON Data Provider

After applying the PTF and performing the upgrade steps, restart the components involved, and then check for expected messages. In particular, check the messages that report the APAR number and Zowe version.

Procedure

1. Stop any running instances of OMEGAMON Data Broker.
2. Stop any running instances of OMEGAMON Data Connect.
3. Use your upgraded method to start OMEGAMON Data Connect.
4. Use your upgraded method to start OMEGAMON Data Broker.
5. Restart your OMEGAMON runtime environments.
6. Check for expected messages.

   In particular, check the following messages for the correct APAR number, OA65724, and the required Zowe version, 2.12.0 or later:

   OMEGAMON Data Broker
   In the SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker (for example, job name KAYS* or ZWES*):

Message	Check
ZWES0001I	Zowe version
KAYB0005I	APAR number

OMEGAMON Data Connect

In the STD0UT file of OMEGAMON Data Connect:

Message	Check
KAYC0035I	APAR number

Related tasks

Starting OMEGAMON Data Provider

Starting OMEGAMON Data Provider involves starting the related components: OMEGAMON Data Connect, OMEGAMON Data Broker, and the data source, such as the OMEGAMON runtime environment.

Related reference

Expected messages

These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

July 2023: APAR OA64880

To upgrade an existing OMEGAMON Data Provider environment to use the PTF for this APAR, you must perform upgrade steps after applying the PTF.

Before you begin

Read what's new in this APAR so that you understand the reasons for these upgrade steps.

Apply the PTF to your z/OS SMP/E target libraries.

Upgrading OMEGAMON Data Connect

Move your OMEGAMON Data Connect configuration files to user directories and upgrade your methods of running OMEGAMON Data Connect.

Procedure

1. Move your existing OMEGAMON Data Connect configuration files to user directories.

   This APAR introduces user directories as a formalized structure for storing your site-specific OMEGAMON Data Connect configuration details.

   Perform the following steps for each of your existing OMEGAMON Data Connect configuration files:

   a) Use the `create` action of the supplied shell script to create a user directory.

      For example, at a shell prompt, change to the `bin` directory under the OMEGAMON Data Connect installation directory, and then enter the following command, where `/var/omdp/prod-a` is the user directory that you want to create:

      ```
      ODP_CONNECT_USER_DIR=/var/omdp/prod-a ./connect create
      ```

   b) Move your existing configuration file to the relative path `config/connect.yaml` under the new user directory. Overwrite the sample configuration file that the script copied to that directory.

2. If you use a z/OS JCL procedure to run OMEGAMON Data Connect, upgrade your procedure based on the latest sample.

a) Replace your procedure with the latest sample procedure supplied in `TKANSAM(KAYCONN)`.

b) Edit the values of the symbolic parameters at the start of the procedure.

JAVAHOME
The path of the installation directory of Java 8, or later, 64-bit edition. The directory must contain a `bin` subdirectory that contains the **`java`** command.

INSTLDIR
The path of the OMEGAMON Data Connect installation directory.

USERDIR
The path of your OMEGAMON Data Connect user directory.

3. If you use the supplied sample shell script to run OMEGAMON Data Connect, change your method for invoking the script to match the latest sample.

   Changes to the sample shell script include:

   - To run OMEGAMON Data Connect, you must now specify the command-line argument `run`.
   - You must set the value of the environment variable `ODP_CONNECT_USER_DIR` to refer to the path of the OMEGAMON Data Connect user directory.

   For more details on the changes to the shell script, see "What's new in OMEGAMON Data Provider" on page 1 and "Sample shell script to run OMEGAMON Data Connect" on page 77.

4. If you use your own custom Java command line to run OMEGAMON Data Connect, upgrade your command line to reflect the changes in OMEGAMON Data Connect.

   In particular, note the new JAR file name in the `-jar` option and the new `-Dodp.ext` option.

   For details on changes to the Java command line, see "What's new in OMEGAMON Data Provider" on page 1 and "Java command line to run OMEGAMON Data Connect" on page 79.

Related concepts

Where and how to run OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Related reference

OMEGAMON Data Connect user directory
An OMEGAMON Data Connect user directory contains files that configure OMEGAMON Data Connect for your site.

OMEGAMON Data Connect installation directory
The OMEGAMON Data Connect installation directory contains the files for OMEGAMON Data Connect that are supplied with OMEGAMON Data Provider.

Upgrading OMEGAMON Data Broker

Upgrade the Zowe cross-memory server that runs OMEGAMON Data Broker.

About this task

These upgrade steps are required for two reasons:

- OMEGAMON Data Broker now requires a Zowe cross-memory server load module from Zowe 1.28.2 or later.
- You are *no longer required* to rename the Zowe cross-memory server load module supplied with OMEGAMON Data Provider from `KAYSIS01` to `ZWESIS01`. OMEGAMON Data Provider supplies the server load module from Zowe 1.28.2. You can use that load module with the name `KAYSIS01`.

If you run OMEGAMON Data Broker in a Zowe cross-memory server that is part of a separate installation of Zowe earlier than 1.28.2, then either:

- Upgrade your separate installation of Zowe to a more recent version. Skip the upgrade procedure described here. Instead, see the Zowe documentation.

  Ensure that the **STEPLIB** concatenation for your Zowe cross-memory server job step includes the latest OMEGAMON Data Broker load modules, supplied in TKANMODP(KAYB0001) and TKANMODP(KAYBNETL).
- Switch to using the server load module, from Zowe 1.28.2, that is supplied with OMEGAMON Data Provider. For details, see the following procedure.

If you already run OMEGAMON Data Broker in a Zowe cross-memory server that is supplied with OMEGAMON Data Provider, then perform the following procedure to replace your existing ZWE-prefix members with KAY-prefix members.

Tip: Performing the following procedure to replace your existing ZWE-prefix members with KAY-prefix members is not strictly necessary. The important point for this upgrade is that, whatever name you use for the Zowe cross-memory server load module, you use a load module from Zowe 1.28.2 or later. However, switching to KAY-prefix members will save you future work and avoids the risk of inadvertently continuing to use an old copy of the server module. OMEGAMON Data Provider supplies the load module as KAYSIS01. If you decide to continue renaming it to ZWESIS01, as required by earlier APARs, then, if you ever forget to perform that rename, you'll continue to use the old renamed copy. It's your choice, but it's less work and less risky to switch to using the KAYSIS01 load module name as supplied with OMEGAMON Data Provider.

Procedure

1. Replace your existing OMEGAMON Data Broker startup JCL procedure, PROCLIB(ZWESIS01), with a new procedure, PROCLIB(KAYSIS01), based on the new sample member TKANSAM(KAYSIS01).

   The new sample procedure refers to the supplied load module name KAYSIS01, reflecting the fact that this load module no longer needs to be renamed to ZWESIS01. The STEPLIB can now refer to the single library, TKANMODP, that contains all of the required load modules.

   The new sample procedure specifies the name ODP_BROKER (was: ZWESIS_STD). The name that you specify here must match the name specified by the `broker.name` key in the collection configuration member, RKANPARU(KAYOPEN). If you decide to keep the name ODP_BROKER, to identify this instance of the Zowe cross-memory server as dedicated to running OMEGAMON Data Broker, then ensure that you also update RKANPARU(KAYOPEN).

   For more details on editing the sample procedure, see "Configuring OMEGAMON Data Broker" on page 54.
2. Rename your existing OMEGAMON Data Broker configuration member to match the first four characters of the server load module name.

   For example, rename the existing member PARMLIB(ZWESIP00) to PARMLIB(KAYSIP00).
3. Define a resource profile for KAYS*.* started tasks to match the existing ZWES*.* profile.

   Example RACF commands:

   ```
   RDEFINE STARTED KAYS*.* STDATA(USER(OMEGSTC) GROUP(STCGROUP)
   PRIVILEGED(NO) TRUSTED(NO))
   SETROPTS RACLIST(STARTED) REFRESH
   ```

   Replace the parameter values in this example with values that reflect your site-specific practices for started tasks.
4. If the resource profile for ZWES*.* started tasks was required *only* for the Zowe cross-memory server that is dedicated to running OMEGAMON Data Broker, then delete that profile.

   Attention: Only delete the ZWES*.* profile if you are certain that it is no longer required. For example, confirm that the profile is not required by a separate Zowe installation unrelated to OMEGAMON Data Provider.

Example RACF commands:

```
RDELETE STARTED ZWES*.*
SETROPTS RACLIST(STARTED) REFRESH
```

5. Modify the z/OS MVS program properties table (PPT) to make the Zowe cross-memory server load module KAYSIS01 run in key 4 and be non-swappable.
   a) Edit the PPT definition member SYS1.PARMLIB(SCHED*xx*).
      Add the following entry:

      ```
      PPT PGMNAME(KAYSIS01) KEY(4) NOSWAP
      ```

   b) Modify the PPT.
      Example MVS system command:

      ```
      SET SCH=xx
      ```

6. If the existing PPT entry for ZWESIS01 was used *only* for the Zowe cross-memory server that is dedicated to running OMEGAMON Data Broker, then delete that entry.

   Attention: Only delete the PPT entry for ZWESIS01 if you are certain that it is no longer required. For example, confirm that the entry is not required by a stand-alone Zowe installation unrelated to OMEGAMON Data Provider.

Related tasks

Configuring OMEGAMON Data Broker
OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. You can configure OMEGAMON Data Broker using either the server that is supplied with OMEGAMON Data Provider (the best and easiest choice for most users) or a server in an existing separate Zowe installation.

Restarting OMEGAMON Data Provider

After applying the PTF and performing the upgrade steps, restart the components involved, and then check for expected messages. In particular, check the messages that report the APAR number and Zowe version.

Procedure

1. Stop any running instances of OMEGAMON Data Broker.
2. Stop any running instances of OMEGAMON Data Connect.
3. Use your upgraded method to start OMEGAMON Data Connect.
4. Use your upgraded method to start OMEGAMON Data Broker.
5. Restart your OMEGAMON runtime environments.
6. Check for expected messages.

   In particular, check the following messages for the correct APAR number, OA64880, and the required Zowe version, 1.28.2 or later:

   OMEGAMON Data Broker
   In the SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker (for example, job name KAYS* or ZWES*):

Message	Check
ZWES0001I	Zowe version
KAYB0005I	APAR number

OMEGAMON Data Connect

In the STD0UT file of OMEGAMON Data Connect:

Message	Check
KAYC0035I	APAR number

Related tasks

Starting OMEGAMON Data Provider

Starting OMEGAMON Data Provider involves starting the related components: OMEGAMON Data Connect, OMEGAMON Data Broker, and the data source, such as the OMEGAMON runtime environment.

Related reference

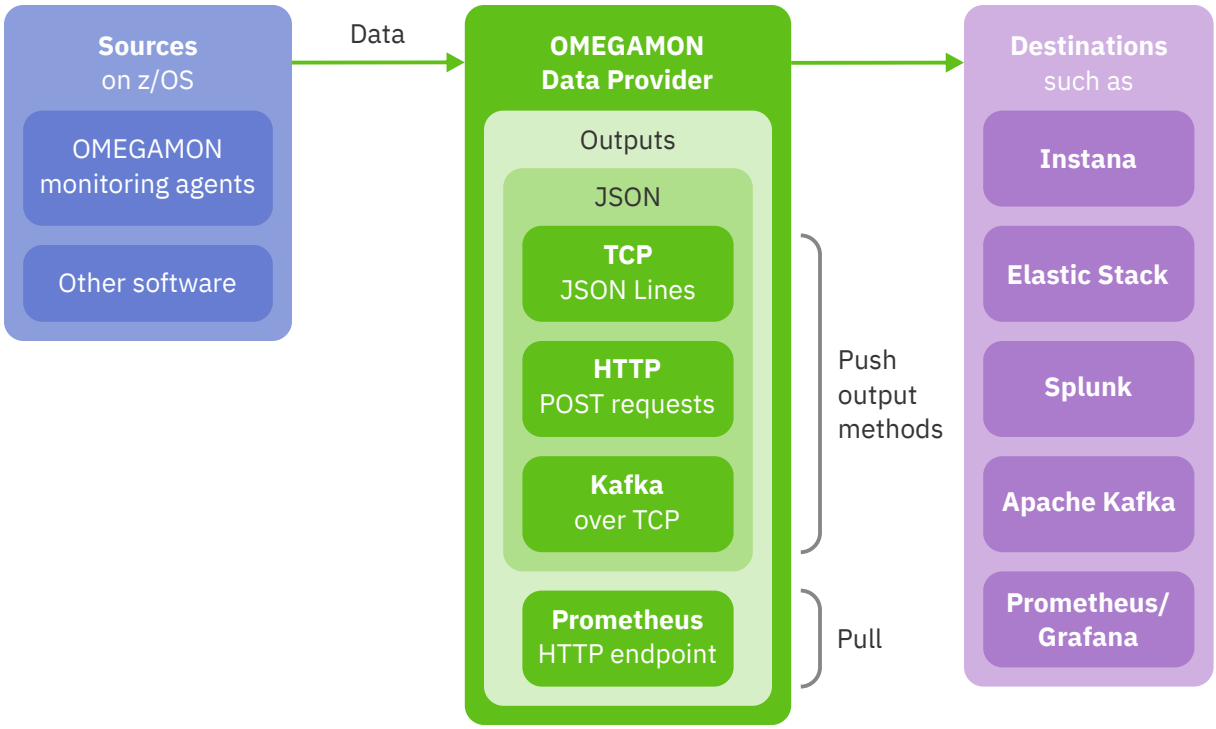
Expected messages

These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Introduction to OMEGAMON Data Provider

OMEGAMON Data Provider makes data from z/OS available to various destinations. Typically, the data consists of metrics or events from z/OS subsystems or applications, and the destinations are analytics platforms that analyze that data.

Figure 1. OMEGAMON Data Provider: data sources, output methods, and example destinations



Data sources

Data sources for OMEGAMON Data Provider include the following software:

- OMEGAMON monitoring agents on z/OS
- Other software on z/OS, such as:
  - IBM Db2 Automation Expert for z/OS
  - IBM Db2 Query Monitor for z/OS

This OMEGAMON Data Provider documentation includes information that applies to all data sources, with specific details for OMEGAMON monitoring agents. For specific details about other software as a data source for OMEGAMON Data Provider, see the separate documentation for that software.

Output methods

Output from OMEGAMON Data Provider is designed to be easily ingested by destinations such as analytics platforms.

OMEGAMON Data Provider supports the following output methods:

Push
The following output methods "push" (send) data in JSON format to destinations that have been configured to listen for this input:

TCP
JSON Lines over TCP

HTTP
HTTP POST requests containing JSON

Kafka
Apache Kafka topics containing JSON

Pull
The following output method enables destinations to "pull" (get, scrape) data from OMEGAMON Data Provider:

Prometheus
Prometheus metrics HTTP endpoint served by OMEGAMON Data Provider

Some output methods are for a specific analytics platform. Other output methods, such as JSON Lines over TCP, are generic, and can be ingested by various analytics platforms. For example, analytics platforms that can ingest JSON Lines over TCP include IBM Instana Observability on z/OS (Instana), the Elastic Stack, and Splunk.

Some analytics platforms support more than one output method from OMEGAMON Data Provider. To help decide which output method to use, see the documentation for your analytics platform.

Example destinations

Destinations for OMEGAMON Data Provider include the following platforms:

- IBM Instana
- Elastic Stack
- Splunk
- Apache Kafka
- Prometheus/Grafana

Related reference
OMEGAMON monitoring agents supported by OMEGAMON Data Provider

OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

OMEGAMON Data Provider architecture

OMEGAMON Data Provider consists of two components: OMEGAMON Data Broker and OMEGAMON Data Connect.

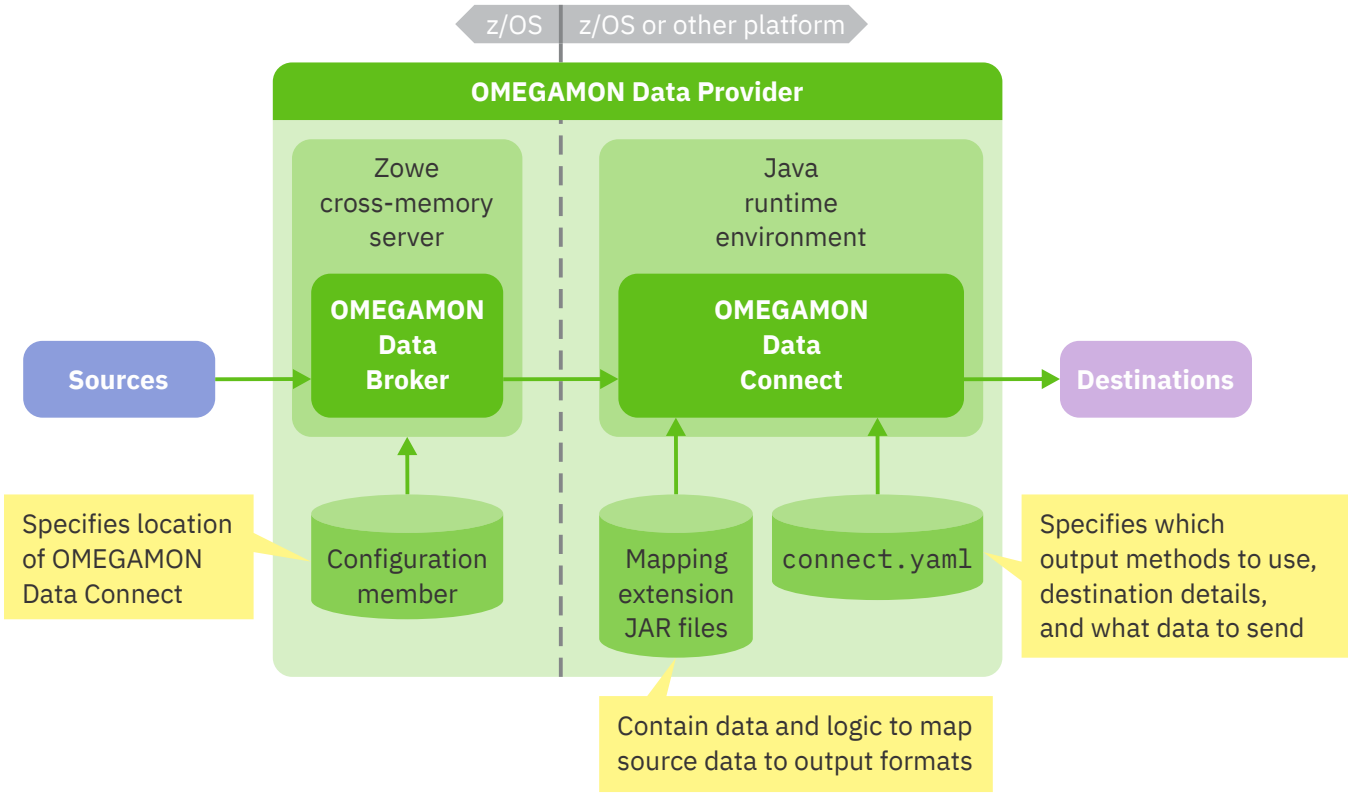


Figure 2. Components of OMEGAMON Data Provider

OMEGAMON Data Broker

OMEGAMON Data Broker forwards data from sources, such as OMEGAMON monitoring agents, over a TCP network to OMEGAMON Data Connect.

OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server.

Tip: *You don't need to install Zowe*. The Zowe cross-memory server is supplied with OMEGAMON Data Provider. For details, see "Prerequisites for OMEGAMON Data Provider" on page 44.

The Zowe cross-memory server runs in its own z/OS address space on the same z/OS instance as the data source.

The behavior of OMEGAMON Data Broker is determined by plain-text key-value configuration parameters in the Zowe cross-memory server configuration member. The parameters include details such as the hostname and port on which OMEGAMON Data Connect is listening for data.

OMEGAMON Data Connect

OMEGAMON Data Connect receives data from OMEGAMON Data Broker, transforms the data from its original format, and publishes the data using one or more output methods.

Each instance of OMEGAMON Data Connect can publish to multiple destinations:

- Multiple TCP destinations ("sinks")
- Multiple HTTP endpoints that accept POST requests

- One Prometheus metrics HTTP endpoint served by OMEGAMON Data Connect
- One Kafka cluster

You can optionally filter which tables and which fields to publish to each destination.

OMEGAMON Data Connect is a Java application developed using the Spring Boot framework. The framework provides features such as application metrics published to Prometheus by Micrometer, which you can use to monitor OMEGAMON Data Connect activity and performance. For details, see the Spring Boot documentation for the Spring Boot Actuator endpoints.

You can run OMEGAMON Data Connect on z/OS or on any platform that meets the Java requirements for OMEGAMON Data Connect.

The behavior of OMEGAMON Data Connect is determined by a YAML document, `config/connect.yaml`, in the OMEGAMON Data Connect user directory of your choice.

OMEGAMON Data Connect is a modular application that consists of multiple Java archive (JAR) files:

Core JAR file
A single JAR file that implements the core functionality of OMEGAMON Data Connect.

OMEGAMON Data Provider supplies the core JAR file in the OMEGAMON Data Connect installation directory in the following path:

`lib/odp-server-`*`version`*`.jar`

Mapping extension JAR files
A JAR file for each data source of OMEGAMON Data Provider.

Mapping extensions extend OMEGAMON Data Connect to support data from different sources. Typically, the source data from z/OS is in a proprietary binary format. Mapping extensions consist of Java classes that contain data and logic to map the source data to the output formats of OMEGAMON Data Connect. Mapping extension Java classes are sometimes referred to as *mapping classes*.

OMEGAMON Data Provider supplies mapping extension JAR files for OMEGAMON monitoring agents in the OMEGAMON Data Connect installation directory in the following path:

`lib/ext/k`*`pp`*`-odp-model-`*`version`*`.jar`

where k*pp* is the product code for the agent.

Other data sources must supply their own mapping extension JAR file. You must copy the mapping extension JAR file from the installation directory of that other software to your OMEGAMON Data Connect user directory.

Related concepts

Where and how to run OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Related reference

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Configuration
OMEGAMON Data Provider involves four configuration points: the data source, the two components of OMEGAMON Data Provider (OMEGAMON Data Broker and OMEGAMON Data Connect), and the destination.

Characteristics of JSON output from OMEGAMON Data Connect
If you need to work directly with JSON output from OMEGAMON Data Connect, then it's useful to understand the characteristics of this data, such as its structure, property names, and property values.

OMEGAMON Data Connect user directory

An OMEGAMON Data Connect user directory contains files that configure OMEGAMON Data Connect for your site.

OMEGAMON Data Provider topology

OMEGAMON Data Provider topology typically consists of one instance of OMEGAMON Data Broker per z/OS LPAR, with multiple instances of OMEGAMON Data Broker feeding a single instance of OMEGAMON Data Connect.

The following figure shows an example topology with multiple instances of OMEGAMON Data Broker sending data to a single instance of OMEGAMON Data Connect:

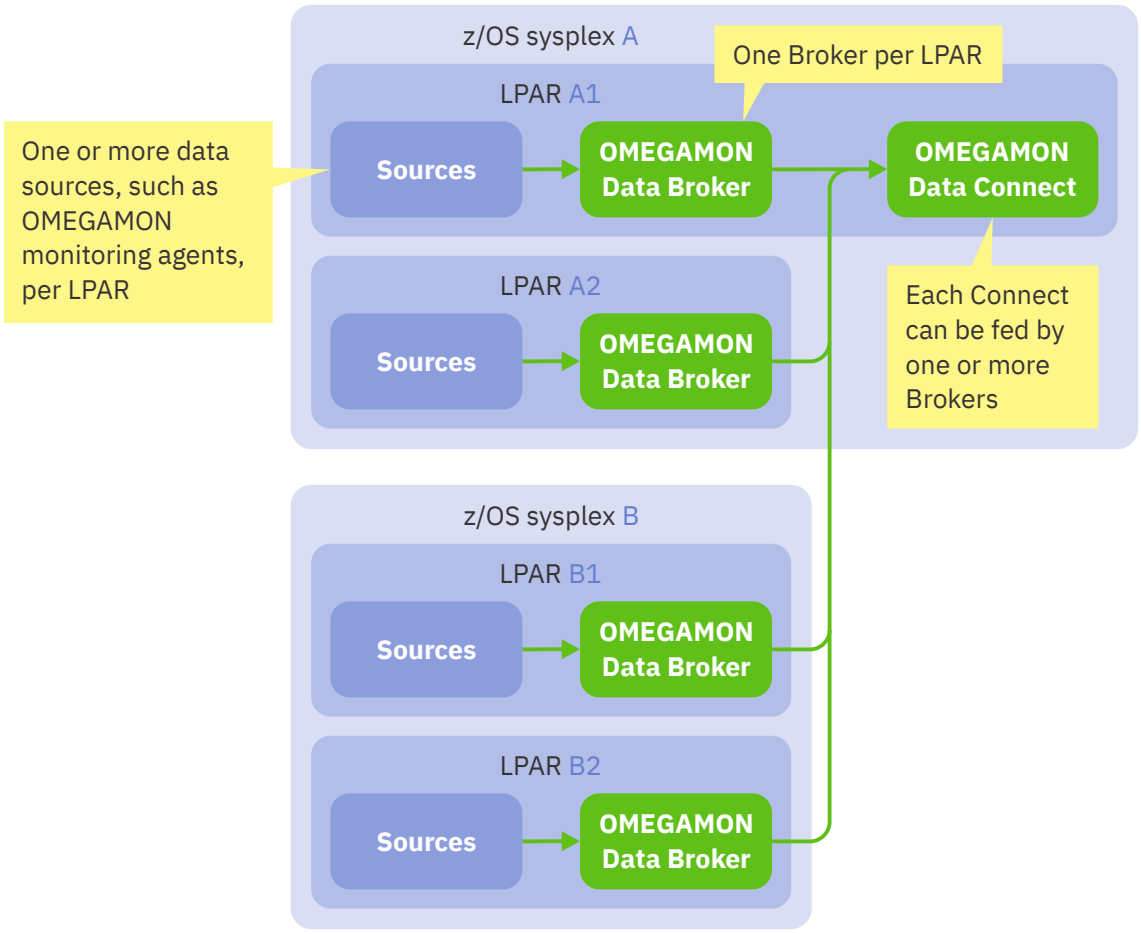


Figure 3. Example OMEGAMON Data Provider topology

In this example, OMEGAMON Data Connect runs in a z/OS LPAR that is also running OMEGAMON Data Broker. Alternatively, OMEGAMON Data Connect can run in a z/OS LPAR that is not running OMEGAMON Data Broker, or OMEGAMON Data Connect can run on a different platform, such as a remote, non-mainframe Linux® system.

OMEGAMON Data Broker can forward to multiple instances of OMEGAMON Data Connect

You can configure each instance of OMEGAMON Data Broker to forward data to multiple instances of OMEGAMON Data Connect.

The following figure shows an example topology where each instance of OMEGAMON Data Broker feeds two instances of OMEGAMON Data Connect:

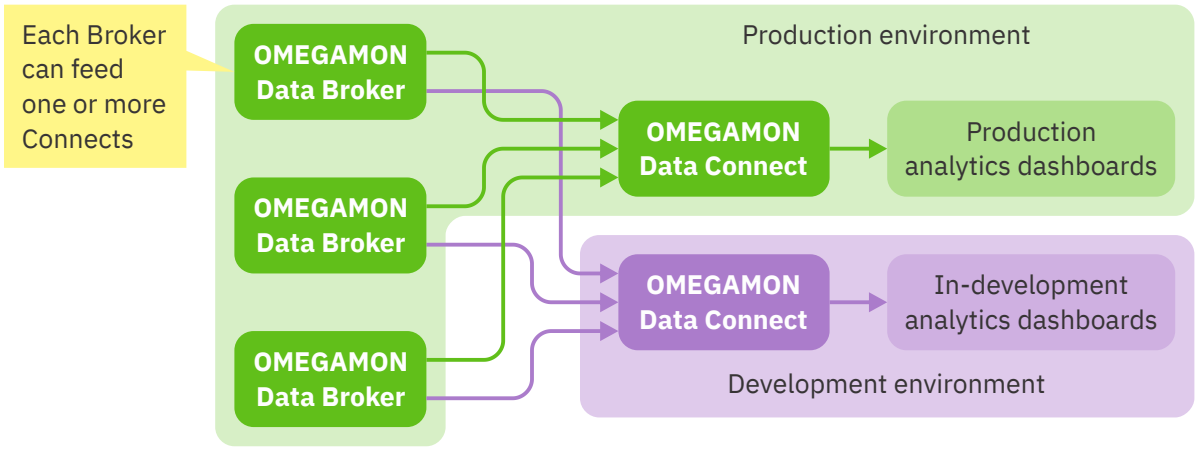


Figure 4. Example topology with separate instances of OMEGAMON Data Connect for production and development

In this example, one instance of OMEGAMON Data Connect belongs to a production environment while the other instance belongs to a development environment. Configuring OMEGAMON Data Broker to feed both instances of OMEGAMON Data Connect has the following advantages:

- You can stop, start, and reconfigure the development instance of OMEGAMON Data Connect without interrupting the flow of data to production analytics dashboards.
- You don't have to drive a separate workload to send data to in-development analytics dashboards. The development environment receives data from the production workload.

Related reference

Configuration
OMEGAMON Data Provider involves four configuration points: the data source, the two components of OMEGAMON Data Provider (OMEGAMON Data Broker and OMEGAMON Data Connect), and the destination.

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

OMEGAMON Data Provider security

You can secure each component of OMEGAMON Data Provider, including their input and output communication methods.

You can use Transport Layer Security (TLS), including HTTPS (HTTP over TLS), to secure output from OMEGAMON Data Provider to external destinations.

The following figure shows the connections between components of OMEGAMON Data Provider, and the options for unsecure versus secure communication protocols.

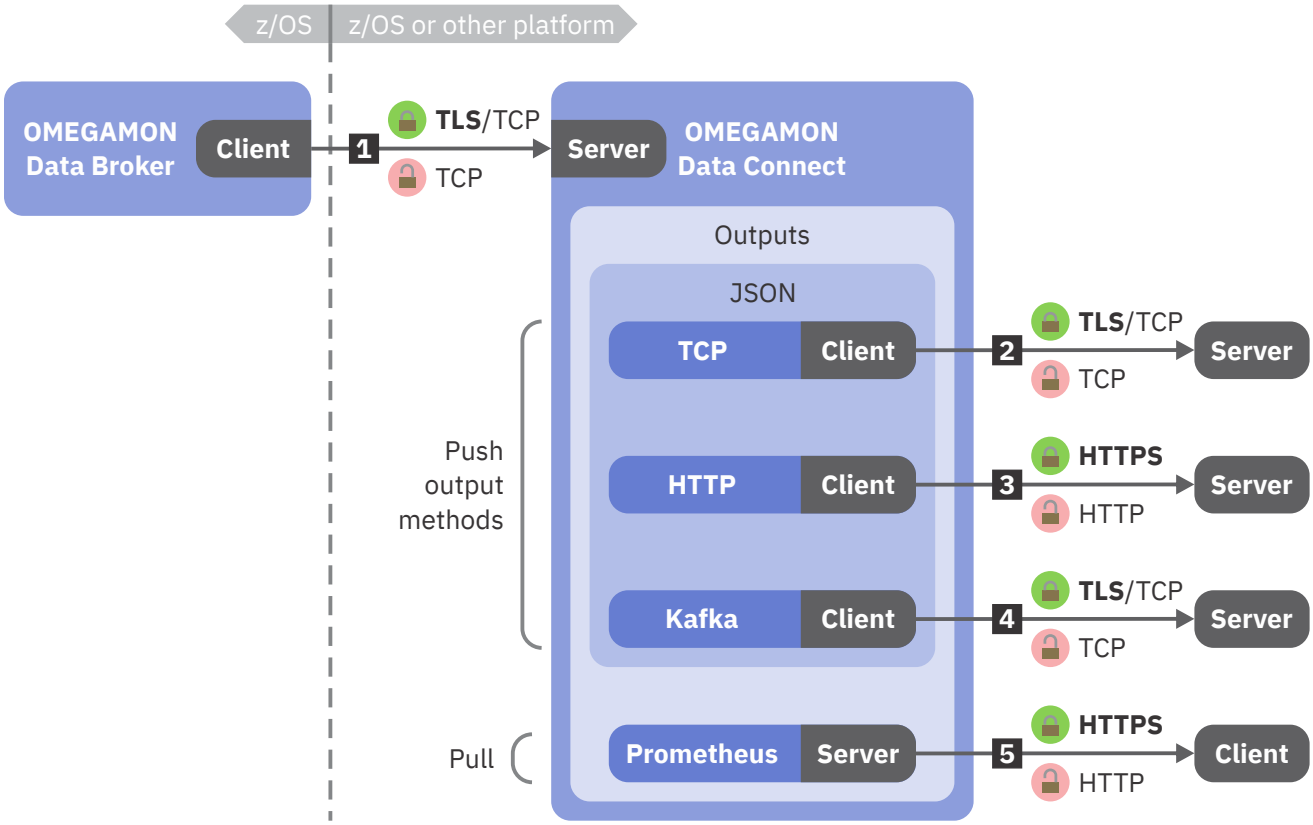


Figure 5. OMEGAMON Data Provider communication protocols with or without TLS

Table 1. Connections between OMEGAMON Data Provider components, with links to security parameter descriptions

Connection	Source	Destination	Secure protocol
1	Client: OMEGAMON Data Broker	Server: OMEGAMON Data Connect TCP input	TLS over TCP
2	Client: OMEGAMON Data Connect TCP output	Server: Analytics platform or application	TLS over TCP
3	Client: OMEGAMON Data Connect HTTP output	Server: Instana acting as an HTTP(S) server	HTTPS
4	Client: OMEGAMON Data Connect Kafka output	Server: Kafka server	TLS over TCP
5	Server: OMEGAMON Data Connect Prometheus output	Client: Prometheus server acting as an HTTP(S) client	HTTPS

To control the permission to run each component of OMEGAMON Data Provider and access to the related data sets, use your system's access control facility. For example, on z/OS, use RACF.

OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider

To use OMEGAMON monitoring agents as a data source, OMEGAMON Data Provider extends OMEGAMON attribute collection.

OMEGAMON attribute collection

To understand how OMEGAMON Data Provider extends OMEGAMON attribute collection, it's useful to understand how OMEGAMON attribute collection works before introducing OMEGAMON Data Provider.

The OMEGAMON family of products includes monitoring agents that collect performance, behavior, and resource usage metrics of systems and applications on z/OS. In OMEGAMON, these metrics are known as *attributes*. Related attributes are organized into *groups*. Attribute groups are also known as *tables*.

To configure OMEGAMON attribute collection, you create *historical collections*. Each historical collection specifies an attribute group that you want to collect and a *collection interval*: a minimum of 1 minute to a maximum of 1 day.

To create historical collections, you use the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli® Enterprise Portal (TEP). For more information about creating historical collections, see the OMEGAMON documentation for e3270UI and TEP.

OMEGAMON stores recently collected attributes, also known as *near-term historical data*, in a set of files known as the persistent data store (PDS).

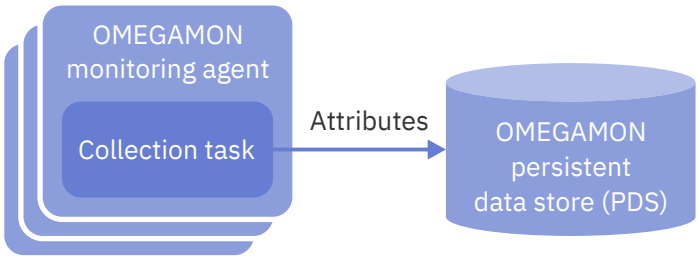


Figure 6. OMEGAMON attribute collection before introducing OMEGAMON Data Provider

How OMEGAMON Data Provider extends OMEGAMON attribute collection

OMEGAMON Data Provider extends OMEGAMON attribute collection by introducing a new runtime environment member, RKANPARU(KAYOPEN), that sets the destination of collected attributes: PDS, OMEGAMON Data Provider, both, or neither.

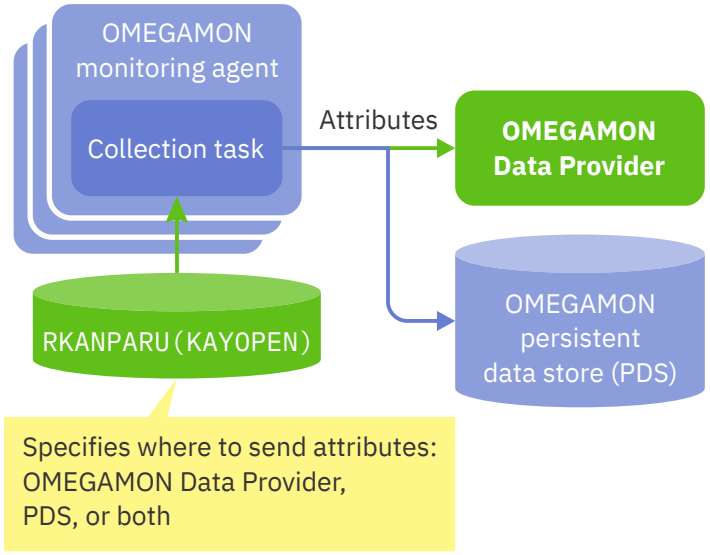


Figure 7. OMEGAMON runtime environment member RKANPARU(KAYOPEN) sets the destination of collected attributes

The prerequisite APAR level of the OMNIMON Base element extends the collection task of OMEGAMON monitoring agents to look for the runtime environment member RKANPARU(KAYOPEN).

If the member does not exist, then OMEGAMON Data Provider is dormant and the collection task stores attributes in the PDS only.

Otherwise, the member is a YAML document that sets the destinations of collected attributes: OMEGAMON Data Provider, PDS, both, or neither.

You can choose to continue storing attributes in the PDS while *also* sending attributes to OMEGAMON Data Provider, or you can choose to *only* send attributes to OMEGAMON Data Provider.

To pass attributes through to OMEGAMON Data Provider without storing them on disk in the PDS, you can specify OMEGAMON Data Provider as the only destination. Sending attributes only to OMEGAMON Data Provider is known as *passthrough*.

If you need to retrieve near-term historical data from the PDS, then you must specify PDS as a destination. For example, if you need to display near-term historical data in the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP), then you must specify PDS as a destination.

OMEGAMON Data Provider publishes attributes as they are collected. The data frequency is determined by the collection interval. For example, suppose you have created a historical collection for an attribute group and set the collection interval to 1 minute. If you configure OMEGAMON Data Provider to send those attributes as JSON Lines over TCP, then the destination analytics platform receives data for that attribute group every minute.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections

and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

Starter dashboards

You can get a set of starter Elastic Kibana dashboards that visualize attributes from OMEGAMON monitoring agents.

The starter dashboards are available from GitHub. For details, see the documentation website.

You can use the starter dashboards as a starting point for analyzing your own data and developing your own dashboards.

If you cannot access GitHub, then, for alternative methods of getting the starter dashboards, contact your IBM Software representative for OMEGAMON products.

Related tasks

Integrating the Elastic Stack with OMEGAMON Data Provider
To integrate the Elastic Stack with OMEGAMON Data Provider, you can configure the OMEGAMON Data Connect component of OMEGAMON Data Provider to send data as JSON Lines over TCP to Logstash. You can configure Logstash to listen on a TCP port for that JSON Lines and forward the data to Elasticsearch.

Prerequisites for OMEGAMON Data Provider

Some prerequisites for OMEGAMON Data Provider are common to all data sources. Other prerequisites are specific to the data source.

This documentation describes prerequisites that are common to all data sources and prerequisites that are specific to OMEGAMON monitoring agents as a data source.

For prerequisites that are specific to other data sources, see the documentation for those data sources.

Related concepts

Starter dashboards
You can get a set of starter Elastic Kibana dashboards that visualize attributes from OMEGAMON monitoring agents.

Related reference

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Prerequisites for all data sources

These prerequisites are common to all data sources for OMEGAMON Data Provider.

A product offering that contains OMEGAMON Data Provider

OMEGAMON Data Provider is not available separately. You must have one of the following product offerings that contain OMEGAMON Data Provider:

Table 2. Product offerings that contain OMEGAMON Data Provider

Product offering	Minimum version that contains OMEGAMON Data Provider
IBM Z Monitoring Suite	1.2.1
IBM Z Service Management Suite	2.1.1
IBM Db2 Management Solution Pack for z/OS	3.1

Table 2. Product offerings that contain OMEGAMON Data Provider (continued)	
Product offering	**Minimum version that contains OMEGAMON Data Provider**
IBM Db2 Performance Solution Pack for z/OS	2.1
IBM zSystems Integration for Observability	6.1
IBM Z OMEGAMON AI for JVM (stand-alone monitoring agent)	6.1
IBM Z OMEGAMON AI for Networks (stand-alone monitoring agent)	6.1
IBM Z OMEGAMON AI for z/OS (stand-alone monitoring agent)	6.1
IBM Db2 Automation Expert for z/OS	1.1
IBM Db2 Query Monitor for z/OS	3.4

For details on how each product offering packages OMEGAMON Data Provider, such as which FMIDs you need to install, see the program directory for each product offering.

z/OS

The z/OS version that OMEGAMON Data Provider requires depends on where you decide to run the OMEGAMON Data Connect component:

- If you decide to run OMEGAMON Data Connect on z/OS, then OMEGAMON Data Provider requires z/OS 2.5 or later.
- Otherwise, OMEGAMON Data Provider requires z/OS 2.4 or later.

The z/OS version that OMEGAMON Data Provider requires is dictated by the distinct requirements of each component of OMEGAMON Data Provider:

- **OMEGAMON Data Broker** is a plug-in for the z/OS-based Zowe cross-memory server. Running OMEGAMON Data Broker using the server that is supplied with OMEGAMON Data Provider **requires z/OS 2.4 or later**. For details, see "Zowe cross-memory server (supplied with OMEGAMON Data Provider)" on page 45.
- **OMEGAMON Data Connect** runs on any platform that supports Java 17 or later, 64-bit edition. On z/OS, Java 17 **requires z/OS 2.5 or later**. For details, see "Java" on page 46.

Each product offering that contains OMEGAMON Data Provider and each data source for OMEGAMON Data Provider has its own prerequisites. Those prerequisites might include different z/OS versions than cited here. For details, see the separate documentation for that software.

Zowe cross-memory server (supplied with OMEGAMON Data Provider)

The OMEGAMON Data Broker component of OMEGAMON Data Provider is a plug-in for the Zowe cross-memory server. However, OMEGAMON Data Provider does not require you to install Zowe.

Instead, OMEGAMON Data Provider supplies the two Zowe parts (load modules) that OMEGAMON Data Broker requires:

Table 3. Zowe parts required by OMEGAMON Data Broker and supplied with OMEGAMON Data Provider		
Description	**Library and member name as supplied with OMEGAMON Data Provider**	**Library and member name as supplied with Zowe**
Zowe cross-memory server load module	`TKANMODP(KAYSIS01)`	`SZWEAUTH(ZWESIS01)`

Table 3. Zowe parts required by OMEGAMON Data Broker and supplied with OMEGAMON Data Provider (continued)

Description	Library and member name as supplied with OMEGAMON Data Provider	Library and member name as supplied with Zowe
ZISDYNAMIC, the ZIS dynamic linkage base plug-in: a Zowe cross-memory server plug-in (load module) that is required by OMEGAMON Data Broker	TKANMODP(KAYSISDL)	SZWEAUTH(ZWESISDL)

OMEGAMON Data Provider has no other dependencies on Zowe.

For most users, the best and easiest choice is to configure OMEGAMON Data Broker using the Zowe cross-memory server that is supplied with OMEGAMON Data Provider. However, if you already have a compatible version of Zowe installed (2.12.0 or any later 2.*x*.*x*), then you can configure OMEGAMON Data Broker using an existing server in that separate Zowe installation.

Java

OMEGAMON Data Connect is a Java application that requires Java 17 or later, 64-bit edition.

Running OMEGAMON Data Connect on z/OS requires z/OS 2.5 or later. Java 17 runtime environments on z/OS require IBM Semeru Runtime Certified Edition for z/OS 17. IBM Semeru Runtime Certified Edition for z/OS 17 requires z/OS 2.5 or later. Java 17 is not available on z/OS 2.4 or earlier.

OMEGAMON Data Provider supplies a sample JCL procedure that runs OMEGAMON Data Connect on z/OS. The sample JCL requires the Java Batch Launcher and Toolkit for z/OS (JZOS). JZOS is supplied with IBM Semeru Runtime Certified Edition for z/OS.

Related concepts

Where and how to run OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Related tasks

Configuring OMEGAMON Data Broker
OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. You can configure OMEGAMON Data Broker using either the server that is supplied with OMEGAMON Data Provider (the best and easiest choice for most users) or a server in an existing separate Zowe installation.

Prerequisites for OMEGAMON monitoring agents as a data source

These prerequisites apply only if you are using OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider.

OMEGAMON monitoring agents

To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you must have one or more OMEGAMON monitoring agents supported by OMEGAMON Data Provider.

OMNIMON Base APAR level

OMNIMON Base, minimum version 7.5.0, APAR/PTF level OA62052/UJ06872, extends the collection task of OMEGAMON monitoring agents to work with OMEGAMON Data Provider. To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you must have at least that minimum APAR level of OMNIMON Base.

All product offerings that include OMEGAMON monitoring agents also include the OMNIMON Base element.

OMEGAMON runtime environment

Configure an OMEGAMON runtime environment that you want to use with OMEGAMON Data Provider.

At a minimum, the runtime environment must include the following items:

- A monitoring server.
- One or more monitoring agents supported by OMEGAMON Data Provider.
- Historical data collection configured to collect at least one attribute group from one of those monitoring agents.

  For example, Address Space CPU Utilization attributes (table name: `ascpuutil`).

  Important: Except for attribute tables that *must* be collected at the TEMS (monitoring server), set the collection location of the historical data collection to TEMA (monitoring agent).

  If you plan to use this runtime environment as a data source for the starter Elastic Kibana dashboards for OMEGAMON Data Provider, then see the separate documentation for those dashboards for details on which tables you need to collect.

Before proceeding, test that the runtime environment successfully collects attributes. For example, view the attribute data in the OMEGAMON enhanced 3270 user interface (e3270UI) or in the Tivoli Enterprise Portal (TEP).

Starter Elastic Kibana dashboards

You can get starter Elastic Kibana dashboards that visualize attributes from OMEGAMON monitoring agents.

The starter dashboards are not a prerequisite. They are an optional, ready-made starting point for analyzing output from OMEGAMON Data Provider. You can process output from OMEGAMON Data Provider with the software of your choice. For example:

- You can use a command shell to display STDOUT output from OMEGAMON Data Provider.
- You can configure your analytics platform to ingest attributes from OMEGAMON Data Provider.
- You can use a command-line TCP listener tool to save JSON Lines from OMEGAMON Data Provider to a file, and then examine the contents of the file in an editor.
- You can develop your own application to process attributes from OMEGAMON Data Provider.

Related tasks

Configuring OMEGAMON monitoring agents as a data source
To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you need to configure the runtime member RKANPARU(KAYOPEN) to specify which historical collections to send to OMEGAMON Data Broker.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

Installing OMEGAMON Data Provider

If you have the prerequisite software, then OMEGAMON Data Provider is already installed in your z/OS SMP/E target libraries. You need to know the location of those libraries. Also, before configuring OMEGAMON Data Provider, you might need to install some components in other locations.

Before you begin

You need to understand the OMEGAMON Data Provider architecture.

Ensure that you have the prerequisite software.

You need to know the SMP/E target locations where OMEGAMON Data Provider is installed. In particular:

- The high-level qualifiers of the TKANMODP and TKANSAM MVS libraries.
- The z/OS UNIX directory path specified by the SMP/E **DDDEF** TKAYHFS. The default path is `/usr/lpp/omdp`.

If you do not know these locations, then contact the person who installed the prerequisite product offering that contains OMEGAMON Data Provider.

About this task

The following procedure ensures that each component is installed in the correct location.

Procedure

1. If you are using OMEGAMON monitoring agents as a data source: ensure that your OMEGAMON runtime environment (RTE) load modules are at the prerequisite software levels for OMEGAMON Data Provider.

   If you have not already done so, follow your site-specific procedures to refresh the runtime members in the RKANMOD* libraries of your RTE from the TKANMOD* SMP/E target libraries. For example, perform the **GENERATE** action of Monitoring Configuration Manager.

   This step ensures that the collection tasks in your RTE support OMEGAMON Data Provider.
2. Decide whether you want to run OMEGAMON Data Connect on or off z/OS.

   The SMP/E installation steps for product offerings that contain OMEGAMON Data Provider create a z/OS UNIX directory that is specified by the **DDDEF** name TKAYHFS. The directory contains OMEGAMON Data Connect. The default directory path is `/usr/lpp/omdp`.

   That z/OS UNIX directory contains one subdirectory and three files:

   kay-110
   The OMEGAMON Data Connect installation directory. This directory contains the expanded files for running OMEGAMON Data Connect.

   KAY11PAX
   A pax interchange format archive file containing the OMEGAMON Data Connect installation directory.

   KAY11SH
   A z/OS UNIX shell script that extracts the pax file KAY11PAX into the subdirectory `kay-110`. This script will already have been run by an SMP/E **APPLY** command when the product offering that contains OMEGAMON Data Provider was installed.

   KAY11ZIP
   A compressed binary file with the same contents as the KAY11PAX file, but using a different compressed file format.

If you want to run OMEGAMON Data Connect on z/OS, but you would prefer to run it from a location that is not an SMP/E target, then copy the installation directory, `kay-110`, to the z/OS UNIX path of your choice.

If you want to run OMEGAMON Data Connect off z/OS:

a. Transfer the compressed binary file `KAY11ZIP` from z/OS UNIX to another platform, such as Linux.

   Consider renaming the transferred copy with a `.zip` file extension.

b. Create the OMEGAMON Data Connect installation directory `kay-110` on that platform by expanding the compressed file.

Results

The software components involved in OMEGAMON Data Provider are now installed in the locations you will run them. Before you run them, you must configure them.

Related concepts

Prerequisites for OMEGAMON Data Provider
Some prerequisites for OMEGAMON Data Provider are common to all data sources. Other prerequisites are specific to the data source.

Where and how to run OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Related reference

OMEGAMON Data Connect installation directory
The OMEGAMON Data Connect installation directory contains the files for OMEGAMON Data Connect that are supplied with OMEGAMON Data Provider.

Getting started with OMEGAMON Data Provider

After installing OMEGAMON Data Provider, you need to configure and then start its components.

About this task

OMEGAMON Data Provider can receive data from multiple sources and send data in multiple formats to multiple destinations.

While the procedures for getting started are similar for all combinations, configuration details can differ depending on the input data source, output format, and destination. The procedures presented here provide details for the following configuration:

Data source	Data format	Destination
OMEGAMON monitoring agents	JSON Lines	An analytics platform, such as the Elastic Stack, that is configured to ingest JSON Lines over TCP

For details on configuring other data sources, see the separate documentation for each data source.

For comprehensive details on configuring each component of OMEGAMON Data Provider, see “Configuration” on page 89.

These procedures configure each OMEGAMON Data Provider component, and then, when all the components have been configured, start the components.

Results

These procedures configure and start OMEGAMON Data Provider on *one* z/OS instance (LPAR).

To use OMEGAMON Data Provider to send data from other LPARs, you need to configure and start OMEGAMON Data Provider on each LPAR. You don't necessarily need to configure and start an instance of OMEGAMON Data Connect for each LPAR. That topology depends on your specific requirements.

Related concepts

OMEGAMON Data Provider architecture
OMEGAMON Data Provider consists of two components: OMEGAMON Data Broker and OMEGAMON Data Connect.

OMEGAMON Data Provider topology
OMEGAMON Data Provider topology typically consists of one instance of OMEGAMON Data Broker per z/OS LPAR, with multiple instances of OMEGAMON Data Broker feeding a single instance of OMEGAMON Data Connect.

Configuring data sources for OMEGAMON Data Provider

You need to configure each data source to send data to OMEGAMON Data Provider.

Procedure

Follow the procedure for each data source.

Data source	Procedure
OMEGAMON monitoring agents	“Configuring OMEGAMON monitoring agents as a data source” on page 52

Data source	Procedure
Others	See the separate documentation for each data source

Configuring OMEGAMON monitoring agents as a data source

To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you need to configure the runtime member RKANPARU(KAYOPEN) to specify which historical collections to send to OMEGAMON Data Broker.

Before you begin

Historical collections are a prerequisite for using OMEGAMON Data Provider. Before configuring OMEGAMON Data Provider, you need to create historical collections.

To create historical collections, you use the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP). For more information about creating historical collections, see the OMEGAMON documentation for e3270UI and TEP.

Some analytics platforms might require, or provide specific support for, particular attributes. You need to create the corresponding historical collections for those attributes. For information about some analytics platforms, see “Integrating analytics platforms with OMEGAMON Data Provider” on page 63. Otherwise, see the documentation for your analytics platform.

You need to know the location of the TKANSAM library installed by the prerequisite OMEGAMON software.

About this task

OMEGAMON Data Provider uses the runtime member RKANPARU(KAYOPEN) to specify destinations for historical collections.

The KAYOPEN member is optional. If you omit it, then collected attributes are sent only to the persistent data store (PDS), not OMEGAMON Data Broker.

KAYOPEN specifies which *attribute groups* (tables) to send to OMEGAMON Data Broker. Later, when configuring OMEGAMON Data Connect, you can specify which *attributes* (fields) to publish from those tables.

The TKANSAM library contains a sample KAYOPEN member.

Procedure

1. Copy the TKANSAM(KAYOPEN) member to a location of your choice where you want to permanently store your primary customized copy of this member.

   Attention: The KAYOPEN member is not managed by PARMGEN or Monitoring Configuration Manager. Some actions of PARMGEN and Monitoring Configuration Manager, such as the **GENERATE** action of Monitoring Configuration Manager, delete RKANPARU library members. PTF UJ93077 for APAR OA64681 (2Q23) excludes members with the name pattern KAY*, such as KAYOPEN, from being deleted. If your site does not yet have that PTF applied, then you must maintain your primary copy of KAYOPEN in a different location of your choice and, after each GENERATE action, copy KAYOPEN to the RKANPARU library.
2. Edit the KAYOPEN member in the permanent location you have chosen to maintain it.

```
broker:
  name: ODP_BROKER          # 1
collections:
  - product: km5            # 2
    table: assumry
    interval: 1
    destination:
      - open
      - pds
  - product: km5
    table: ascpuutil
    interval: 1
    destination:
      - open
      - pds
```

Figure 8. Excerpt of sample OMEGAMON Data Provider collection configuration member, KAY0PEN

1

The `broker.name` key (the `name` child key of the `broker` key) refers to the name of the Zowe cross-memory server that runs OMEGAMON Data Broker.

The name is the value of the **NAME** runtime parameter of the JCL **EXEC** statement that runs the server.

Tip: Use the value ODP_BROKER as specified in the sample configuration member. That value matches the name in the sample JCL member to run the server, described later in "Configuring OMEGAMON Data Broker" on page 54.

2

For each existing historical collection that you want to send to OMEGAMON Data Broker, insert a corresponding entry under the `collections` key to select the collection.

Each entry must specify the following selection criteria:

`product`
The 3-character k*pp* product code of the monitoring agent that owns the table.

`table`
The table name.

`interval`
The collection interval in minutes or the special value 0 (zero).

The value 0 acts as a wildcard; it selects all historical collections for the table, regardless of collection interval.

If you specify a number of minutes, ensure that the value matches the interval of a historical collection that you want to send.

`destination`
To send attributes to OMEGAMON Data Broker, the destinations must include the value open.

For comprehensive details on editing KAY0PEN, see "Configuration parameters for OMEGAMON monitoring agents as a data source" on page 89.

3. Copy the customized KAY0PEN member from its permanent location to the RKANPARU library of your OMEGAMON runtime environment.
4. If the monitoring agents that own the tables for the collections are running, then reload the collection configuration in each of the affected agents.

Related concepts

Prerequisites for OMEGAMON monitoring agents as a data source

These prerequisites apply only if you are using OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider.

Related tasks

Reloading OMEGAMON collection configuration
After updating collection configuration parameters in the RKANPARU(KAYOPEN) member, you need to apply the configuration changes to the affected collection tasks. To apply the changes, you can either restart the jobs that run the collection tasks, or enter the MVS **MODIFY** system command presented here.

Adding more OMEGAMON collections to OMEGAMON Data Provider
If you have already configured an OMEGAMON runtime environment to send collections to OMEGAMON Data Provider, then follow the steps here to add more.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Configuring OMEGAMON Data Broker

OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. You can configure OMEGAMON Data Broker using either the server that is supplied with OMEGAMON Data Provider (the best and easiest choice for most users) or a server in an existing separate Zowe installation.

Before you begin

If you do not have an existing separate Zowe installation, skip to "Configuring OMEGAMON Data Broker using the server supplied with OMEGAMON Data Provider" on page 56.

Otherwise, decide whether to use the server that is supplied with OMEGAMON Data Provider or a server in a separate Zowe installation. The following tables describe advantages and disadvantages of each option.

Tip: Using the server supplied with OMEGAMON Data Provider is the best and easiest choice for most users. Using an existing server in a separate Zowe installation introduces compatibility, configuration, and maintenance issues. If you're content to accept this tip without reading more, skip to "Configuring OMEGAMON Data Broker using the server supplied with OMEGAMON Data Provider" on page 56.

Table 4. Advantages of using the server supplied with OMEGAMON Data Provider

Advantages of using the server supplied with OMEGAMON Data Provider	**Corresponding disadvantages of using an existing server in a separate Zowe installation**
You don't need to install Zowe. Instead, OMEGAMON Data Provider supplies the few Zowe parts required to run OMEGAMON Data Broker.	Installing Zowe involves more effort than using the parts that are supplied with OMEGAMON Data Provider. A Zowe installation involves many parts that OMEGAMON Data Provider does not use.
Guaranteed compatibility between the server and OMEGAMON Data Broker.	Compatibility issues: • You must check that the Zowe version is compatible with OMEGAMON Data Broker: currently, Zowe 2.12.0 or any later 2.*x*.*x*. • You must reconsider compatibility whenever you update either OMEGAMON Data Provider or your separate Zowe installation.

Table 4. Advantages of using the server supplied with OMEGAMON Data Provider (continued)	
Advantages of using the server supplied with OMEGAMON Data Provider	**Corresponding disadvantages of using an existing server in a separate Zowe installation**
OMEGAMON Data Provider supplies the load modules for the server and OMEGAMON Data Broker in a single library, TKANMODP. The **STEPLIB** for the server JCL can refer to that single library, without relocating any members.	Configuration issue: • You must decide how to make OMEGAMON Data Broker available to the server; how to include the required load modules in the **STEPLIB** for the server JCL. Maintenance issue: • You must manage how to apply updates to the OMEGAMON Data Broker load modules.
Using a dedicated server for OMEGAMON Data Broker enables you to manage the configuration, maintenance, and availability of OMEGAMON Data Provider independently of a separate Zowe installation. **Tip:** You can run multiple instances of the Zowe cross-memory server on the same z/OS instance. A dedicated server for OMEGAMON Data Broker can coexist on the same z/OS instance with a server in a separate Zowe installation.	Maintenance issue: • You must manage the effects of updates to the Zowe installation on OMEGAMON Data Broker.

Table 5. Advantages of using an existing server in a separate Zowe installation	
Advantages of using an existing server in a separate Zowe installation	**Corresponding disadvantages of using the server supplied with OMEGAMON Data Provider**
Minimizes the number of address spaces on z/OS. For example, if you are already running compatible Zowe cross-memory servers in the z/OS instances where you need to run OMEGAMON Data Broker, and you are comfortable managing the compatibility, configuration, and maintenance issues, then you can host OMEGAMON Data Broker in those existing servers rather than using dedicated servers for OMEGAMON Data Broker.	Requires a new address space.
Uses existing PROCLIB and PARMLIB (JCL procedure and configuration) members.	Requires you to create a JCL procedure, a configuration member, and a corresponding started task.

If you need more information to help you decide which option to choose, review the corresponding configuration procedures for each option.

Procedure

Follow the procedure for configuring OMEGAMON Data Broker using your chosen option.

Related concepts

Prerequisites for all data sources
These prerequisites are common to all data sources for OMEGAMON Data Provider.

Related reference

OMEGAMON Data Broker configuration parameters

OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

Configuring OMEGAMON Data Broker using the server supplied with OMEGAMON Data Provider

Configuring OMEGAMON Data Broker using the server supplied with OMEGAMON Data Provider is the best and easiest choice for most users.

Before you begin

You need to know the location of the TKANMODP and TKANSAM libraries that were installed by the product offering that contains OMEGAMON Data Provider.

About this task

OMEGAMON Data Broker involves the following load modules, all of which are supplied in the TKANMODP library:

TKANMODP(KAYSIS01)
Zowe cross-memory server. The same load module is supplied with Zowe as SZWEAUTH(ZWESIS01).

TKANMODP(KAYSISDL)
ZISDYNAMIC, the dynamic linkage base plug-in for the Zowe cross-memory server. This plug-in is required by OMEGAMON Data Broker. The same load module is supplied with Zowe as SZWEAUTH(ZWESISDL).

TKANMODP(KAYB0001)
OMEGAMON Data Broker, a plug-in for the Zowe cross-memory server.

TKANMODP(KAYBNETL)
A load module used by OMEGAMON Data Broker.

Configuring OMEGAMON Data Broker involves the following sample members supplied in the TKANSAM library:

TKANSAM(KAYSIS01)
JCL procedure that runs the Zowe cross-memory server. This member is required only if you decide to configure a new server.

TKANSAM(KAYSIP00)
Zowe cross-memory server configuration member that contains OMEGAMON Data Broker configuration parameters.

Procedure

1. Modify the z/OS MVS program properties table (PPT) to make the Zowe cross-memory server load module KAYSIS01 run in key 4 and be non-swappable.
   a) Edit the PPT definition member SYS1.PARMLIB(SCHED*xx*).
      Add the following entry:

      ```
      PPT PGMNAME(KAYSIS01) KEY(4) NOSWAP
      ```

   b) Modify the PPT.
      Example MVS system command:

      ```
      SET SCH=xx
      ```

2. Ensure that the TKANMODP library is APF-authorized.

   Depending on which product offering you installed that contains OMEGAMON Data Provider, and how you configured that software, the TKANMODP library might already be APF-authorized.

To check whether the library is APF-authorized, enter the following MVS system command:

```
D PROG,APF,DSNAME=omegamon_hlq.TKANMODP
```

where *omegamon_hlq* represents the high-level qualifiers of the TKANMODP library.

To dynamically add the SMS-managed library to the APF list, enter:

```
SETPROG APF,ADD,DSNAME=omegamon_hlq.TKANMODP,SMS
```

3. Copy the sample Zowe cross-memory server startup JCL procedure TKANSAM(KAYSIS01) to your choice of PROCLIB library. For example, your site-specific library for user procedures, or the system procedure library, SYS1.PROCLIB.
4. Edit the new copy of the JCL procedure, PROCLIB(KAYSIS01).

```
//KAYSIS01  PROC NAME='ODP_BROKER',MEM=00,RGN=0M     1
…
//BROKER    EXEC PGM=KAYSIS01,REGION=&RGN,
//            PARM='NAME=&NAME,MEM=&MEM'
//STEPLIB   DD   DSNAME=omegamon.TKANMODP,DISP=SHR   2
//PARMLIB   DD   DSNAME=SYS1.PARMLIB,DISP=SHR        3
//SYSPRINT  DD   SYSOUT=*
```

Figure 9. JCL procedure that starts the Zowe cross-memory server, PROCLIB(KAYSIS01)

1

The supplied sample JCL procedure specifies the name of the Zowe cross-memory server as ODP_BROKER. You only need to change the name if you run more than one instance of the server on the same instance of z/OS.

The MEM parameter specifies the last two characters of the Zowe cross-memory server PARMLIB member name, *pppp*IP*xx*, where *pppp* is the first four characters of the Zowe cross-memory server load module name. The default MEM value is 00. Given the server load module name KAYSIS01 and MEM value 00, then the PARMLIB member name matches the supplied sample, KAYSIP00. You only need to change the MEM value if you have different versions of this member in the same PARMLIB library.

2

Edit the **STEPLIB DD** statement to refer to the TKANMODP library on your system.

Note:

- Data sets in this **STEPLIB** concatenation must be a partitioned data set extended (PDSE), not a PDS. TKANMODP is a PDSE.
- Data sets in this **STEPLIB** concatenation must be APF-authorized.
- Use a **STEPLIB DD** statement to identify the location of the Zowe cross-memory server load library, so that the job refers to the appropriate specific version of the software. Do not add the load library to the system LNKLST or LPALST.
- The Zowe cross-memory server loads itself into the link pack area (LPA) so that it can use PC-cp services (program call in the user's primary address space).

3

Either edit the **PARMLIB DD** statement to refer to the library containing the Zowe cross-memory server configuration member or, if you copied that member to the system PARMLIB, remove this statement. The low-level qualifier PARMLIB in the example dsname reflects a typical convention, not a requirement. It is your choice where to store this configuration member.

5. Define a resource profile to control the security of the started task that will use the PROCLIB(KAYSIS01) procedure.

The user that you associate with the started task must have an OMVS segment.

Example RACF commands:

```
RDEFINE STARTED KAYS*.* STDATA(USER(OMEGSTC) GROUP(STCGROUP)
PRIVILEGED(NO) TRUSTED(NO))
SETROPTS RACLIST(STARTED) REFRESH
```

Replace the parameter values in this example with values that reflect your site-specific practices for started tasks.

6. Copy the sample Zowe cross-memory server configuration member `TKANSAM(KAYSIP00)` to your choice of `PARMLIB` library. For example, your site-specific library for user parameter members, or the system parameter library, `SYS1.PARMLIB`.
7. Edit the new copy of the configuration member, `PARMLIB(KAYSIP00)`.

   The following listing shows a minimal complete set of required parameters:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

KAY.CIDB.FWD=ON
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK_HOST=<connect_hostname>
KAY.CIDB.FWD.FWD1.SINK_PORT=<connect_port>
```

   Set the values of the following parameters:

   KAY.CIDB.FWD.*<forwarder_id>*.SINK_HOST=*<connect_hostname>*
   Hostname or IP address of the OMEGAMON Data Connect instance that is listening for data from OMEGAMON Data Broker.

   In the context of the OMEGAMON Data Broker forwarder, OMEGAMON Data Connect is a *sink*: a destination.

   If you plan to run OMEGAMON Data Connect on the same z/OS instance as OMEGAMON Data Broker, then you can specify the value `localhost` or the local loopback IP address. The typical local loopback IPv4 address is `127.0.0.1`.

   KAY.CIDB.FWD.*<forwarder_id>*.SINK_PORT=*<connect_port>*
   The port on which OMEGAMON Data Connect is listening. Follow your site-specific standards for assigning port numbers.

   For details of other OMEGAMON Data Broker parameters, such as parameters for a secure (SSL/TLS) connection to OMEGAMON Data Connect, see "OMEGAMON Data Broker configuration parameters" on page 95.

Results

Later, when you start the Zowe cross-memory server, the server will load the OMEGAMON Data Broker plug-in, and OMEGAMON Data Broker will connect to OMEGAMON Data Connect.

Configuring OMEGAMON Data Broker using an existing server in a separate Zowe installation

Only configure OMEGAMON Data Broker using an existing server if you understand the advantages and disadvantages. Using an existing server introduces compatibility, configuration, and maintenance issues that do not occur if you use the server supplied with OMEGAMON Data Provider.

Before you begin

Check that the version of your Zowe installation is compatible with OMEGAMON Data Broker. OMEGAMON Data Broker requires a server from Zowe 2.12.0 or any later 2.*x*.*x*.

You need to know the location of the TKANMODP and TKANSAM libraries that were installed by the product offering that contains OMEGAMON Data Provider.

You need to know the locations of the PROCLIB and PARMLIB (JCL procedure and configuration) members for the existing server.

About this task

OMEGAMON Data Broker involves the following load modules in the Zowe installation:

SZWEAUTH(ZWESIS01)
Zowe cross-memory server. The same load module is supplied with OMEGAMON Data Provider as TKANMODP(KAYSIS01).

SZWEAUTH(ZWESISDL)
ZISDYNAMIC, the dynamic linkage base plug-in for the Zowe cross-memory server. This plug-in is required by OMEGAMON Data Broker. The same load module is supplied with OMEGAMON Data Provider as TKANMODP(KAYSISDL).

and the following load modules supplied with OMEGAMON Data Provider:

TKANMODP(KAYB0001)
OMEGAMON Data Broker, a plug-in for the Zowe cross-memory server.

TKANMODP(KAYBNETL)
A load module used by OMEGAMON Data Broker.

Procedure

1. Make the OMEGAMON Data Broker load modules supplied with OMEGAMON Data Provider, TKANMODP(KAYB0001) and TKANMODP(KAYBNETL), available to the existing Zowe cross-memory server.

   It is your choice *how* to make these load modules available to the server.

   Some example methods:

   - Copy the KAYB0001 and KAYBNETL members from the TKANMODP library to the Zowe library for server plug-ins that is specified in the **STEPLIB** concatenation for the server.

     For details on the location of the Zowe library for storing plug-ins for the Zowe cross-memory server, see the Zowe documentation.

     This method introduces repetitive maintenance tasks.

     If updates to OMEGAMON Data Provider include new versions of these members, then you must copy the new versions to the Zowe library for server plug-ins.
   - Add the TKANMODP library to the **STEPLIB** concatenation for the server.

     The TKANMODP library must be APF-authorized.

Attention: If updates to Zowe involve changes to the server JCL, then remember to preserve the change that adds TKANMODP to the **STEPLIB**. Otherwise, these OMEGAMON Data Broker load modules will no longer be available to the server.

If you use this method, then, to manage the introduction of changes to the OMEGAMON Data Broker load modules, refer to a *copy* of the TKANMODP library in a location of your choice, not the original SMP/E target location.

2. Ensure that the server configuration member, PARMLIB(ZWESIP*xx*), registers the ZISDYNAMIC plug-in load module.

   Add the following line to the member, if it does not already exist:

   ```
   ZWES.PLUGIN.ZISDYNAMIC=ZWESISDL
   ```

   Important: Only register the ZISDYNAMIC plug-in once. For example, do not attempt to register the ZISDYNAMIC plug-in once referring to the ZWESISDL load module supplied with Zowe, and then again referring to the KAYSISDL load module supplied with OMEGAMON Data Provider. You do not need to copy the TKANMODP(KAYSISDL) member supplied with OMEGAMON Data Provider to the **STEPLIB** concatenation for the server in your separate installation of Zowe. If you use a server in a separate installation of Zowe, refer to the ZWESISDL plug-in load module supplied with Zowe.

3. Append OMEGAMON Data Broker parameters to the server configuration member, PARMLIB(ZWESIP*xx*).

   The following listing shows a minimal set of required OMEGAMON Data Broker parameters:

   ```
   ZWES.PLUGIN.KAY.CIDB=KAYB0001

   KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

   KAY.CIDB.FWD=ON
   KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
   KAY.CIDB.FWD.FWD1.SINK_HOST=<connect_hostname>
   KAY.CIDB.FWD.FWD1.SINK_PORT=<connect_port>
   ```

   A similar example set of OMEGAMON Data Broker parameters is supplied in the sample member TKANSAM(KAYSIP00).

   Set the values of the following parameters:

   KAY.CIDB.FWD.*<forwarder_id>*.SINK_HOST=*<connect_hostname>*
   Hostname or IP address of the OMEGAMON Data Connect instance that is listening for data from OMEGAMON Data Broker.

   In the context of the OMEGAMON Data Broker forwarder, OMEGAMON Data Connect is a *sink*: a destination.

   If you plan to run OMEGAMON Data Connect on the same z/OS instance as OMEGAMON Data Broker, then you can specify the value localhost or the local loopback IP address. The typical local loopback IPv4 address is 127.0.0.1.

   KAY.CIDB.FWD.*<forwarder_id>*.SINK_PORT=*<connect_port>*
   The port on which OMEGAMON Data Connect is listening. Follow your site-specific standards for assigning port numbers.

   For details of other OMEGAMON Data Broker parameters, such as parameters for a secure (SSL/TLS) connection to OMEGAMON Data Connect, see “OMEGAMON Data Broker configuration parameters” on page 95.

   Tip: Don't restart the server yet with the updated configuration. We'll restart it later, after you have configured all components.

Results

Later, when you restart the Zowe cross-memory server, it will load the OMEGAMON Data Broker plug-in, and OMEGAMON Data Broker will connect to OMEGAMON Data Connect.

Configuring OMEGAMON Data Connect

OMEGAMON Data Connect is a Java application that can run on or off z/OS.

Before you begin

You need to know the paths of the following directories:

- OMEGAMON Data Connect installation directory. The default installation directory name is `kay-110`.
- Java runtime environment that you plan to use to run OMEGAMON Data Connect: Java 17 or later, 64-bit edition.

Decide the location of your OMEGAMON Data Connect user directories: where you want to keep your site-specific configuration files.

Procedure

1. Create an OMEGAMON Data Connect user directory.

   The user directory will contain your site-specific OMEGAMON Data Connect configuration file. The location of the user directory is your choice.

   To create a user directory, use the `create` action of the supplied shell script.

   For example, at a shell prompt, change to the `bin` directory under the OMEGAMON Data Connect installation directory, and then enter the following command, where `/var/omdp/prod-a` is the user directory that you want to create:

   ```
   ODP_CONNECT_USER_DIR=/var/omdp/prod-a ./connect create
   ```

2. Edit the `config/connect.yaml` configuration file in the user directory to match your site-specific requirements.

   Example:

   ```
   connect:
     input:
       tcp:
         enabled: true
         hostname: <connect_host>          # 1
         port: <connect_port>              # 2
     output:
       tcp:
         enabled: true
         sinks:
           logstash: # Your choice of sink name (not a fixed key name)
             enabled: true
             hostname: <logstash_host>     # 3
             port: <logstash_port>         # 4
   ```

   1

   Hostname or IP address on which the OMEGAMON Data Connect host listens for data from OMEGAMON Data Broker.

   Some typical values:

   `0.0.0.0`
   All IPv4 addresses on the local machine.

`localhost` or `127.0.0.1`
Loopback address.

This hostname or IP address must correspond to the hostname or IP address to which OMEGAMON Data Broker sends data, specified by the OMEGAMON Data Broker parameter `SINK_HOST`.

If you run OMEGAMON Data Connect and OMEGAMON Data Broker on the same z/OS instance (LPAR), then you can specify `localhost` for both this hostname and the OMEGAMON Data Broker sink hostname.

2
Port on which to listen for data from OMEGAMON Data Broker.

This value must match the OMEGAMON Data Broker parameter `SINK_PORT`.

3
Destination hostname or IP address. For example, the host running Elastic Logstash.

4
Port on which the destination host is listening for JSON Lines over TCP.

The previous example configures OMEGAMON Data Connect to send JSON Lines over TCP. For comprehensive details on configuring OMEGAMON Data Connect, including configuration for other data formats and destinations, see "OMEGAMON Data Connect configuration parameters" on page 113.

Some analytics platforms might require, or provide specific support for, particular types of data. You might choose to filter the output from OMEGAMON Data Connect to send only that data. For information about some analytics platforms, see "Integrating analytics platforms with OMEGAMON Data Provider" on page 63. Otherwise, see the documentation for your analytics platform.

3. If you want to run OMEGAMON Data Connect on z/OS as a started task, use the sample JCL procedure as a starting point.
   a) Copy the sample JCL procedure `TKANSAM(KAYCONN)` (also supplied as `sample/KAYCONN` in the OMEGAMON Data Connect installation directory) to your choice of MVS `PROCLIB` library. For example, `SYS1.PROCLIB`.
   b) Edit the values of the symbolic parameters at the start of the `PROCLIB(KAYCONN)` procedure.

      `JAVAHOME`
      The path of the installation directory of Java 17 or later, 64-bit edition. The directory must contain a `bin` subdirectory that contains the **`java`** command.

      `INSTLDIR`
      The path of the OMEGAMON Data Connect installation directory.

      `USERDIR`
      The path of your OMEGAMON Data Connect user directory.

   c) Define a resource profile to control the security of the started task.

      The user that you associate with the started task must have an OMVS segment.

      Example RACF commands:

      ```
      RDEFINE STARTED KAYCONN.* STDATA(USER(OMEGSTC) GROUP(STCGROUP)
      PRIVILEGED(NO) TRUSTED(NO))
      SETROPTS RACLIST(STARTED) REFRESH
      ```

      Replace the parameter values in this example with values that reflect your site-specific practices for started tasks.

Tip: For ad hoc testing of OMEGAMON Data Connect from a z/OS UNIX shell, use the sample shell script `bin/connect`.

Related concepts
Where and how to run OMEGAMON Data Connect

OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Related reference

OMEGAMON Data Connect configuration parameters
OMEGAMON Data Connect configuration parameters identify inputs, such as the TCP port on which to listen for data from OMEGAMON Data Broker, and outputs, such as destination analytics platforms. You can filter which data to output.

Integrating analytics platforms with OMEGAMON Data Provider

Use the information provided here together with the product documentation for your analytics platform.

Integrating Instana with OMEGAMON Data Provider

To integrate Instana with OMEGAMON Data Provider, you configure the OMEGAMON Data Connect component of OMEGAMON Data Provider to send data to Instana using one of two output methods: HTTP or TCP.

Optionally, you can also configure OMEGAMON Data Provider to send just the data that Instana specifically supports, such as particular OMEGAMON attribute groups.

This documentation provides an overview of configuring OMEGAMON Data Provider to send data to Instana. For additional hints and tips, including details of the corresponding prerequisite OMEGAMON historical data collections, see the Instana configuration topics on GitHub.

For details of the Instana architecture and how to configure Instana to ingest data from OMEGAMON Data Provider, see the IBM Instana Observability on z/OS documentation.

Overview of Instana integration with OMEGAMON Data Provider

Instana can ingest data from two output methods of OMEGAMON Data Provider:

TCP
OMEGAMON Data Provider sends data in JSON Lines format over TCP to a port on which Instana is listening.

HTTP
OMEGAMON Data Provider sends data in JSON format in HTTP POST requests to an Instana HTTP endpoint.

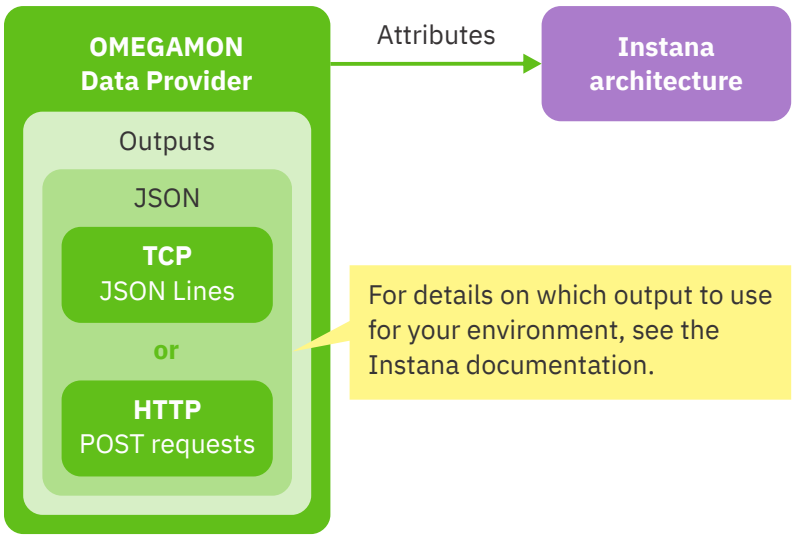


Figure 10. Instana can ingest data from OMEGAMON Data Provider as JSON Lines over TCP or as JSON in HTTP POST requests

For details on which output to use for your environment, see the Instana documentation.

OMEGAMON historical data collections

Historical data collections are a prerequisite for using OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider. Before configuring OMEGAMON Data Provider, you need to create a historical collection for each attribute group supported by Instana.

For details of the attribute groups supported by Instana, see the Instana documentation.

To create historical collections, you use the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP). For more information about creating historical collections, see the OMEGAMON documentation for e3270UI and TEP.

OMEGAMON Data Provider configuration members

Two configuration members specify which OMEGAMON attributes OMEGAMON Data Provider sends, and to where.

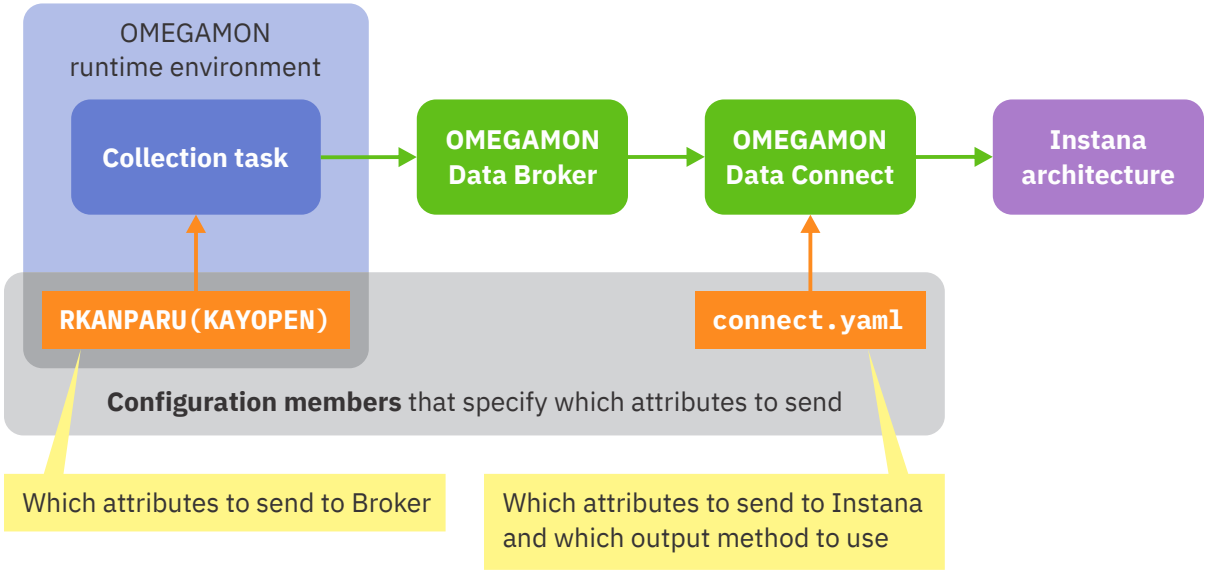


Figure 11. Configuring which OMEGAMON attributes OMEGAMON Data Provider sends, and to where

RKANPARU(KAYOPEN)
Specifies which attribute groups to send to OMEGAMON Data Broker.

OMEGAMON Data Broker forwards attributes to OMEGAMON Data Connect.

The "getting started" task “Configuring OMEGAMON monitoring agents as a data source” on page 52 describes editing the KAYOPEN member.

For KAYOPEN entries that match the attribute groups supported by Instana, see the Instana documentation.

connect.yaml
Specifies where OMEGAMON Data Connect sends attributes, such as Instana.

The "getting started" task “Configuring OMEGAMON Data Connect” on page 61 describes editing `connect.yaml`. When you reach that step, specify a TCP output or an HTTP output for Instana.

Here is the typical format of a TCP output for Instana:

```
connect:
  output:
    tcp:
      enabled: true
```

```
      sinks: # One or more sinks (destinations)
        instana: # Each sink has a unique name of your choice
          enabled: true
          hostname: <instana_host>
          port: <instana_port>
```

Here is the typical format of an HTTP output for Instana, with an HTTPS URL, corresponding SSL/TLS configuration parameters, compressed request bodies, and multiple records per request (batching):

```
connect:
  output:
    http:
      enabled: true
      endpoints: # One or more endpoints (destinations)
        instana: # Each endpoint has a unique name of your choice
          enabled: true
          url: https://<instana_endpoint>
          ssl:
            enabled: true
            protocol: TLSv1.2
            enabled-protocols: TLSv1.2
            key-store: <jks_file_path>
            key-store-type: JKS
            key-store-password: <password>
            trust-store: <jks_file_path>
            trust-store-type: JKS
            trust-store-password: <password>
          compression: true
          batching:
            enabled: true
```

These are typical examples only. For details of the specific configuration required for your environment, such as SSL/TLS, see the Instana documentation.

Integrating the Elastic Stack with OMEGAMON Data Provider

To integrate the Elastic Stack with OMEGAMON Data Provider, you can configure the OMEGAMON Data Connect component of OMEGAMON Data Provider to send data as JSON Lines over TCP to Logstash. You can configure Logstash to listen on a TCP port for that JSON Lines and forward the data to Elasticsearch.

About this task

The information provided here assumes that you are familiar with the Elastic Stack, that you know how to configure the Elastic Stack to ingest data, and that you want to configure your own existing instance of the Elastic Stack.

If you are new to the Elastic Stack, then instead of using the information here, consider using the information provided with the starter dashboards as a starting point.

Related concepts

Starter dashboards

You can get a set of starter Elastic Kibana dashboards that visualize attributes from OMEGAMON monitoring agents.

Related reference

TCP output parameters

OMEGAMON Data Connect TCP output parameters specify one or more destinations ("sinks") for sending data in JSON Lines format over a TCP network.

Basic Elastic Stack configuration for OMEGAMON Data Provider

To ingest JSON Lines from OMEGAMON Data Connect into the Elastic Stack, you need to define a Logstash configuration and an Elasticsearch index template.

The following Elastic Stack configuration defines a minimal basic configuration for ingesting JSON Lines over TCP from OMEGAMON Data Connect.

The configuration described here is a minimal subset of the more detailed configuration for the starter Kibana dashboards.

Elasticsearch configuration

By default, Elasticsearch maps incoming string fields to the **`text`** data type. Elasticsearch parses the contents of `text` fields into tokens for full-text search. You might not want that default behavior for OMEGAMON attribute string values. Many OMEGAMON string fields are names or identifiers. It makes more sense to search these fields as whole values, so the **`keyword`** data type is a better choice. You can configure Elasticsearch by creating an index template that maps string fields to the `keyword` data type.

The result of this mapping is no `.raw` fields. Instead, you use the original field names for sorting and aggregation, because the fields have been mapped to the `keyword` data type.

For example, you can use the following JSON as the body of an Elasticsearch create index template API request:

```
{
  "index_patterns": ["omegamon-*"],
  "template": {
    "settings": {
      "lifecycle": {
        "name": "omegamon-ds-ilm-policy"
      }
    },
    "mappings": {
      "dynamic_templates": [ {
        "strings": {
          "match_mapping_type": "string",
          "mapping": {
            "type": "keyword"
          }
        }
      } ]
    }
  },
  "data_stream": { }
}
```

Figure 12. Elasticsearch index template that maps string fields to the `keyword` data type

Set the `index_patterns` key value to match your site practices for Elasticsearch index names.

Set the `lifecycle.name` to the Elasticsearch index lifecycle policy that you want to use for this data.

The presence of the `data_stream` object in the index template enables data streams.

This example is for use with the `_index_template` API endpoint for *composable* index templates, not the endpoint for deprecated *legacy* index templates.

Logstash pipeline configuration

The following Logstash config listens on a TCP port for JSON Lines from OMEGAMON Data Connect.

```
input {
  tcp {
    id => "omegamon_tcp_input"
    port => 15046
    codec => json_lines
  }
}
filter {
  date {
    match => ["write_time", "ISO8601"]
  }
}
output {
  elasticsearch {
    id => "elasticsearch"
    hosts => ["elasticsearch:9200"]
    index => "omegamon-%{product_code}-%{table_name}-ds"
    action => "create"
    manage_template => false
  }
}
```

Figure 13. Logstash pipeline configuration to ingest JSON Lines over TCP from OMEGAMON Data Connect

Set the `port` on which Logstash listens for input to match the `connect.output.tcp.sinks.`*`sink_name`*`.port` configuration parameter of OMEGAMON Data Connect.

Set the `index` option to match your site practices for Elasticsearch index names.

This example sets the `action` option to `create`, for use with data streams.

Related reference

Fields introduced by OMEGAMON Data Connect
OMEGAMON Data Connect introduces fields that do not correspond to OMEGAMON attributes.

Integrating Splunk with OMEGAMON Data Provider

To integrate Splunk with OMEGAMON Data Provider, you can configure the OMEGAMON Data Connect component of OMEGAMON Data Provider to send data as JSON Lines to a Splunk TCP input.

Related reference

TCP output parameters
OMEGAMON Data Connect TCP output parameters specify one or more destinations ("sinks") for sending data in JSON Lines format over a TCP network.

Basic Splunk configuration for OMEGAMON Data Provider

To ingest JSON Lines from OMEGAMON Data Connect into Splunk, you need to define a Splunk source type that breaks each input line into a separate event, identifies the data format as JSON, and recognizes timestamps. To ingest the data over TCP, you need to define a Splunk TCP input that refers to that source type.

The following Splunk configuration stanzas define a minimal basic configuration for ingesting JSON Lines over TCP from OMEGAMON Data Connect: one stanza in `props.conf`, and one in `inputs.conf`.

Depending on your own site practices, you might perform additional configuration, such as assigning different source types, routing events to different indexes, or using secure TCP (TLS).

Location of Splunk configuration stanzas

This OMEGAMON Data Provider documentation refers to Splunk configuration (`.conf`) file names, but not directory paths. It is your decision where to store the Splunk configuration stanzas for OMEGAMON Data Provider.

For example, you might choose to create a Splunk application directory named *your-organization*-omegamon specifically for OMEGAMON Data Provider, and save the configuration files there:

`$SPLUNK_HOME/etc/apps/`*your-organization*`-omegamon/local/*.conf`

`props.conf`

The following stanza in `props.conf` defines the properties of an "omegamon" source type:

```
[omegamon]
SHOULD_LINEMERGE = false
KV_MODE = json
TIME_PREFIX = \"write_time\":\"
TIME_FORMAT = %Y-%m-%dT%H:%M:%S.%6N%:z
```

The combination of `SHOULD_LINEMERGE = false` and `KV_MODE = json` defines the incoming data as JSON Lines: one event per line, data in JSON format. These two settings apply to different stages in the Splunk data pipeline: **`SHOULD_LINEMERGE`** applies to parsing, before indexing; **`KV_MODE`** applies later, to search-time field extraction.

The regular expression for `TIME_PREFIX` is case sensitive; it matches the lowercase field name `write_time`, which is the field name for event timestamps in JSON from OMEGAMON Data Connect.

The value of `TIME_FORMAT` matches the format of timestamps in JSON from OMEGAMON Data Connect: ISO 8601 date and time of day representation extended format with a zone designator.

`inputs.conf`

The following stanza in `inputs.conf` defines an unsecure TCP input that listens on port 5046, assigns the source type "omegamon" to all incoming events, and stores the events in the default index (typically, main):

```
[tcp://:5046]
sourcetype = omegamon
```

The port number and source type shown here are examples only. The actual values are your choice.

If you have a file of JSON Lines from OMEGAMON Data Connect, then you don't need to define a TCP input. Instead, you can use the Splunk Web **Add Data** > **Upload** option to ingest the file directly from your computer. If you use that technique, remember to select the "omegamon" source type, so that Splunk correctly interprets the file contents.

Tip: In the **Source type** dropdown list on the **Set Source Type** page, the "omegamon" source type will appear under the heading "Uncategorized".

Setting source type per-event based on product code and table name

Rather than assigning the same source type to all events from OMEGAMON Data Connect, you might prefer more granularity; more source types. The method presented here sets the source type per-event based on the values of the JSON keys `product_code` and `table_name`.

You can use transforms in Splunk to override the source type per event.

Each event, each line of JSON Lines, from OMEGAMON Data Connect contains the keys `product_code` and `table_name`. You can use the values of these keys to set the Splunk source type of each event.

Depending on your own site practices, you might perform additional configuration, such as assigning different source types, routing events to different indexes, or using secure TCP.

For example, in `props.conf`, append the following line to the stanza for the corresponding source type or input:

```
TRANSFORMS-changesourcetype = set_sourcetype_omegamon
```

and add the following stanza to `transforms.conf`:

```
[set_sourcetype_omegamon]
# Set sourcetype using values of product_code and table_name fields
REGEX = (?=.*\"product_code\":\"([^\"]+)\")(?=.*\"table_name\":\"([^\"]+)\")
FORMAT = sourcetype::omegamon_$1_$2
DEST_KEY = MetaData:Sourcetype
```

Related reference

Fields introduced by OMEGAMON Data Connect

OMEGAMON Data Connect introduces fields that do not correspond to OMEGAMON attributes.

Starting OMEGAMON Data Provider

Starting OMEGAMON Data Provider involves starting the related components: OMEGAMON Data Connect, OMEGAMON Data Broker, and the data source, such as the OMEGAMON runtime environment.

Before you begin

You should have already configured and started an analytics platform, such as the Elastic Stack, or an application or tool to listen for data from OMEGAMON Data Connect. For example, if you have configured OMEGAMON Data Connect to send JSON Lines over TCP, then you should have tested that the analytics platform successfully ingests JSON Lines over TCP on the specified port. That software should be actively listening now.

If you are using OMEGAMON monitoring agents as a data source, then you should have already tested that your OMEGAMON runtime environment collects attributes in the persistent data store (PDS) *without* OMEGAMON Data Provider. This confirms that you have successfully configured historical data collection, which is a prerequisite for using OMEGAMON Data Provider.

About this task

You can start the components in any order. You don't need to ensure that any components are stopped before you begin. However, the behavior of components and the messages that they issue can depend on the order in which you start them.

The following procedure assumes that the following components are stopped, inactive:

- OMEGAMON Data Connect.
- OMEGAMON Data Broker.

  If you have configured an existing instance of the Zowe cross-memory server to run OMEGAMON Data Broker, that's okay; there's no need to stop it. We'll restart it in the following procedure.

For the purpose of describing a set of expected messages, to help new users, the following procedure starts components in order from "downstream" to "upstream":

1. OMEGAMON Data Connect
2. OMEGAMON Data Broker
3. Data source, such as an OMEGAMON runtime environment

After starting each component, the procedure includes steps to check for expected messages before starting the next component.

If you decide to start the components in a different order, that's okay. Just be aware that the messages issued might differ from the messages described in the following procedure.

Procedure

1. Start OMEGAMON Data Connect.
   - If you have chosen to run OMEGAMON Data Connect on z/OS, here is an example z/OS MVS **START** system command that you can enter to start the OMEGAMON Data Connect started task:

     ```
     S KAYCONN
     ```

   - If you have chosen to run OMEGAMON Data Connect off z/OS, use your platform-specific method to start OMEGAMON Data Connect. For example, the following shell command line runs the supplied sample shell script:

     ```
     ODP_CONNECT_USER_DIR=/var/omdp/prod-a ./connect run
     ```

     – This example shows a single command line that sets the environment variable `ODP_CONNECT_USER_DIR` and then runs the script. Instead of setting the variable each time you run the script, consider using an **`export`** shell command to set the variable in your user profile.
     – The dot and slash (`./`) preceding the script name `connect` assumes that the script is in the current directory. That is, the current directory is the `bin` directory under the OMEGAMON Data Connect installation directory.
     – Replace `/var/omdp/prod-a` with your user directory path.
2. Check the KAYC-prefix messages in the STDOUT output file from OMEGAMON Data Connect.

   You should see several KAYC-prefix messages, including, not necessarily in this order:

   KAYC0023I Starting TCP input service listening on *host*:*port*
   ...
   KAYC0011I Connected to TCP sink: *sink_name* {host: *hostname*, port: *port*}

   Message KAYC0023I indicates that OMEGAMON Data Connect is listening on a TCP port for data from OMEGAMON Data Broker.

   KAYC0011I indicates that OMEGAMON Data Connect has successfully connected to an analytics platform or application that is listening for data on a TCP port.
3. Start the Zowe cross-memory server that runs OMEGAMON Data Broker. If you are using an existing server, stop and then restart the server.

   Example MVS command to start the corresponding started task:

   ```
   S KAYSIS01,REUSASID=YES
   ```

   Zowe cross-memory server supports reusable address spaces and can be started with the `REUSASID=YES` parameter.
4. Check the KAYB-prefix messages in the SYSPRINT output data set of the Zowe cross-memory server job.

   You should see several KAYB-prefix messages, including:

   KAYB0036I Forwarder *forwarder_id* has connected to sink *host*:*port*

   Message KAYB0036I indicates that OMEGAMON Data Broker has connected to the TCP port on which OMEGAMON Data Connect is listening.
5. Start the data source.

   For example, if you are using OMEGAMON monitoring agents as a data source, then start the OMEGAMON runtime environment, if it is not already running. Use your site-specific procedures to start the runtime environment jobs.
6. Check for the expected messages from the data source.

For example, if you are using OMEGAMON monitoring agents as a data source, then check the KPQH-prefix messages in the RKLVLOG output data set of the monitoring agent jobs.

Note: The z/OS and storage monitoring agents run in the same address space as the monitoring server (default job name: OMEGDS).

You should see several KPQH-prefix messages, including:

```
KPQH038I KPQHSMGR: TABLE product.table_name HAS BEEN CONNECTED TO BROKER
```

Message KPQH038I indicates the first time that the collection task sends data for this table to OMEGAMON Data Broker. The timing of this message depends on the collection interval for the table.

7. Check again the KAY-prefix messages in the STDOUT output file from OMEGAMON Data Connect.

   You should see new KAYC-prefix messages:

   ```
   KAYC0008I Creating mapping class for table table_name
   KAYC0033I Table table_name received from origin_type origin_name
   ```

   Message KAYC0008I indicates the first time since starting that this instance of OMEGAMON Data Connect has received data for this table.

   KAYC0033I indicates the first time since starting that this instance of OMEGAMON Data Connect has received data for this table from this *origin_name*.

8. View the attributes in the destination analytics platform or application.

   For example, view the attributes in the starter Elastic Kibana dashboards.

9. Configure and start OMEGAMON Data Provider on other z/OS LPARs.

Related reference

Expected messages

These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Related information

OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

Where and how to run OMEGAMON Data Connect

OMEGAMON Data Connect is a Java application. You can run OMEGAMON Data Connect anywhere that you can run Java 17 or later, 64-bit edition.

Candidate platforms include operating systems such as z/OS UNIX and Linux. Choose a platform that meets your site-specific requirements and preferences. For example, you might prefer to perform *all* OMEGAMON Data Provider processing, including running OMEGAMON Data Connect, on z/OS. Or, to conserve z/OS resources, you might prefer to run OMEGAMON Data Connect off z/OS.

How to run OMEGAMON Data Connect is also your choice, and can depend on the platform. For example:

- On z/OS, JCL
- Shell script
- Java command line

Related concepts

OMEGAMON Data Provider architecture
OMEGAMON Data Provider consists of two components: OMEGAMON Data Broker and OMEGAMON Data Connect.

Prerequisites for all data sources
These prerequisites are common to all data sources for OMEGAMON Data Provider.

Related tasks

Configuring OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application that can run on or off z/OS.

OMEGAMON Data Connect installation directory

The OMEGAMON Data Connect installation directory contains the files for OMEGAMON Data Connect that are supplied with OMEGAMON Data Provider.

If you're responsible for installing, configuring, or running OMEGAMON Data Connect, it's useful to know the location and contents of the installation directory.

Location

The *original* location of the installation directory is a z/OS UNIX directory. The z/OS SMP/E installation steps for OMEGAMON Data Provider create a z/OS UNIX directory specified by the **DDDEF** name TKAYHFS. That directory is the *SMP/E target*. The default path of the SMP/E target directory is `/usr/lpp/omdp`.

The installation directory, `kay-110`, is a child of the SMP/E target.

You can copy the installation directory to a location of your choice, including off z/OS.

Contents

The following tree diagram shows the structure and contents of the installation directory:

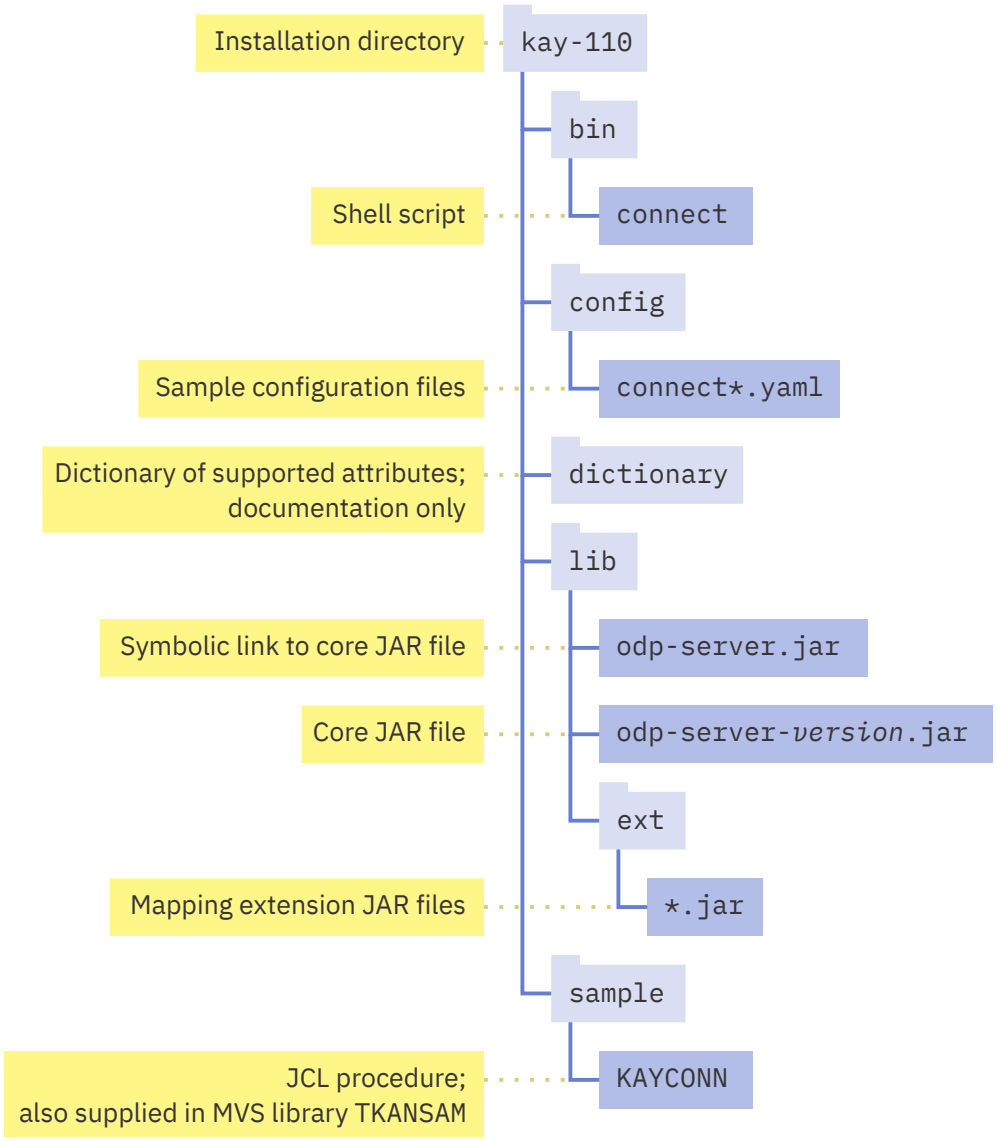


Figure 14. OMEGAMON Data Connect installation directory

The installation directory contains the following files:

- Java archive (JAR) files:
  - Core JAR file
  - Mapping extension JAR files
- Attribute dictionary (documentation)
- Sample configuration files
- Sample files for running OMEGAMON Data Connect:
  - Shell script
  - JCL procedure

Tip: Do not edit sample files in their supplied location in the installation directory; especially, not in their original location under the SMP/E target z/OS UNIX directory. Instead, to avoid updates to the supplied samples overwriting your changes, copy the samples out of the installation directory before editing them. In particular, store your site-specific configuration file in a user directory.

Related tasks

Installing OMEGAMON Data Provider

If you have the prerequisite software, then OMEGAMON Data Provider is already installed in your z/OS SMP/E target libraries. You need to know the location of those libraries. Also, before configuring OMEGAMON Data Provider, you might need to install some components in other locations.

Related reference

OMEGAMON Data Connect user directory
An OMEGAMON Data Connect user directory contains files that configure OMEGAMON Data Connect for your site.

OMEGAMON attribute dictionary
OMEGAMON Data Connect includes a dictionary of OMEGAMON attributes in a set of YAML files.

OMEGAMON Data Connect user directory

An OMEGAMON Data Connect user directory contains files that configure OMEGAMON Data Connect for your site.

If you're responsible for configuring or running OMEGAMON Data Connect, you need to know the location of the user directory and understand its contents.

Location

The location of the user directory is your choice.

Depending on your site-specific topology, you might have multiple OMEGAMON Data Connect user directories, each with a different configuration file.

You might choose to store all OMEGAMON Data Connect user directories under a parent directory such as `/var/omdp`, and then name each user directory according to your site-specific convention. For example, names that reflect your site's system topology: `dev-a`, `dev-b`, `test-a`, `prod-a`.

To create a user directory, use the `create` action of the supplied shell script.

Contents

The following tree diagram shows the structure and contents of a user directory:

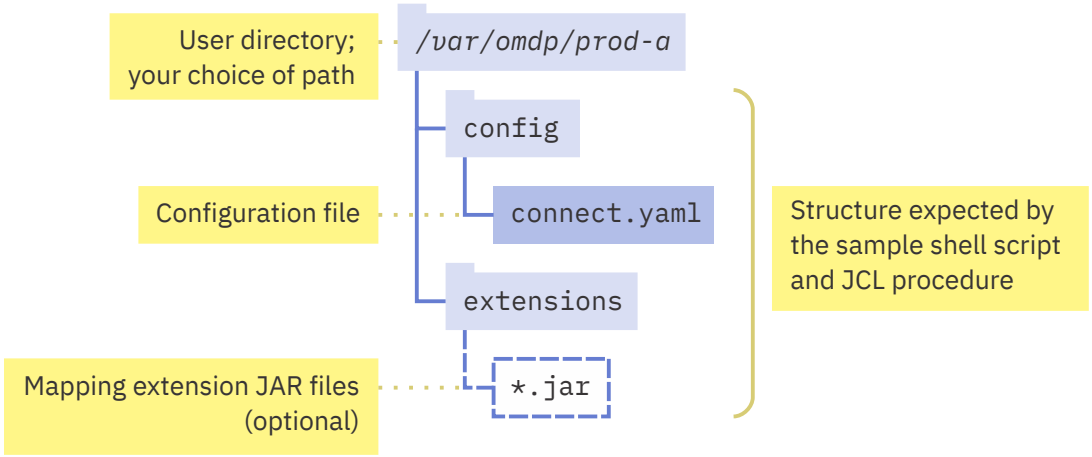


Figure 15. OMEGAMON Data Connect user directory

A user directory can contain the following files:

- Required: a configuration file.
- Optional: mapping extension JAR files.

Storing mapping extension JAR files in a user directory

The OMEGAMON Data Connect installation directory contains mapping extension JAR files for OMEGAMON monitoring agents.

You can also store mapping extension JAR files in a user directory. For example:

- Mapping extension JAR files for data sources other than OMEGAMON monitoring agents.
- If you receive a service update that contains a mapping extension JAR file for an OMEGAMON monitoring agent, you might choose to test that update in a single instance of OMEGAMON Data Connect that refers to a specific user directory, rather than immediately adding the mapping extension JAR file to an installation directory that is used by many instances of OMEGAMON Data Connect.

Related concepts

OMEGAMON Data Provider architecture
OMEGAMON Data Provider consists of two components: OMEGAMON Data Broker and OMEGAMON Data Connect.

Related reference

OMEGAMON Data Connect installation directory
The OMEGAMON Data Connect installation directory contains the files for OMEGAMON Data Connect that are supplied with OMEGAMON Data Provider.

OMEGAMON Data Connect configuration parameters
OMEGAMON Data Connect configuration parameters identify inputs, such as the TCP port on which to listen for data from OMEGAMON Data Broker, and outputs, such as destination analytics platforms. You can filter which data to output.

Sample shell script to run OMEGAMON Data Connect
You can use a shell script to run OMEGAMON Data Connect on operating systems such as z/OS UNIX and Linux. In addition to running OMEGAMON Data Connect, the supplied sample script can also create a user directory containing a sample OMEGAMON Data Connect configuration file as a starting point for you to edit.

Sample JCL procedure to run OMEGAMON Data Connect

You can use JCL to run OMEGAMON Data Connect on z/OS.

OMEGAMON Data Provider supplies a sample JCL procedure that runs OMEGAMON Data Connect on z/OS. The sample JCL requires the Java Batch Launcher and Toolkit for z/OS (JZOS). JZOS is supplied with IBM Semeru Runtime Certified Edition for z/OS.

Location

OMEGAMON Data Provider supplies the sample JCL procedure in two locations:

- MVS library member `TKANSAM(KAYCONN)`
- Under the OMEGAMON Data Connect installation directory in the relative file path:

  `sample/KAYCONN`

  Example absolute file path:

  `/usr/lpp/omdp/kay-110/sample/KAYCONN`

Before running the JCL

The sample JCL procedure requires a user directory that contains your site-specific OMEGAMON Data Connect configuration file.

Before running the JCL:

1. Use the `create` action of the supplied shell script to create a user directory.

2. Edit the `config/connect.yaml` configuration file in the user directory to match your site-specific requirements.

Usage

The sample JCL procedure is designed to be used as a starting point for you to copy and edit to meet your site-specific requirements.

Copy the sample from its supplied location to your choice of MVS `PROCLIB` library. For example, `SYS1.PROCLIB`.

Tip: Review the entire sample so that you understand it completely before using it.

Typically, to edit the procedure for your own use, you only need to edit the values of the following symbolic parameters, near the start of the procedure. You *must* edit the values of these parameters in the **SET** statements in the procedure itself. The sample JCL procedure is *not* sensitive to values for these parameters passed by a calling job or **START** command.

`JAVAHOME`
The path of the installation directory of Java 17 or later, 64-bit edition. The directory must contain a `bin` subdirectory that contains the **`java`** command.

`INSTLDIR`
The path of the OMEGAMON Data Connect installation directory.

`USERDIR`
The path of your OMEGAMON Data Connect user directory.

Related reference

Java command line to run OMEGAMON Data Connect
Whichever platform you choose, you can use a Java command line to run OMEGAMON Data Connect.

Sample shell script to run OMEGAMON Data Connect
You can use a shell script to run OMEGAMON Data Connect on operating systems such as z/OS UNIX and Linux. In addition to running OMEGAMON Data Connect, the supplied sample script can also create a user directory containing a sample OMEGAMON Data Connect configuration file as a starting point for you to edit.

Sample shell script to run OMEGAMON Data Connect

You can use a shell script to run OMEGAMON Data Connect on operating systems such as z/OS UNIX and Linux. In addition to running OMEGAMON Data Connect, the supplied sample script can also create a user directory containing a sample OMEGAMON Data Connect configuration file as a starting point for you to edit.

Location

The sample shell script is supplied under the OMEGAMON Data Connect installation directory in the relative file path:

`bin/connect`

The script file name `connect` has no file extension (no trailing `.sh`).

Example absolute file path:

`/usr/lpp/omdp/kay-110/bin/connect`

Tip: Run the script from its supplied location in the installation directory. Don't copy the script to a different directory and try to run it from there. The script uses its own path to locate files in the installation directory. If you try to run the script from a different directory, the script won't find those files. If you want to make the script available in a directory that is already listed in your PATH environment variable, then, in that other directory, create a symbolic link that refers to the script in the installation directory. Running the symbolic link will run the script in the installation directory.

Syntax

```
connect action
```

The script requires an *action* argument: `create` or `run`.

The script refers to the value of the environment variable `ODP_CONNECT_USER_DIR`. Before calling the script, set the variable to the path of a user directory: depending on the action, either a directory that you want to create or an existing directory that you want to use to run OMEGAMON Data Connect.

`create` action

```
connect create
```

The `create` action creates an OMEGAMON Data Connect user directory.

Before performing the `create` action, set the environment variable `ODP_CONNECT_USER_DIR` to the path of the user directory that you want to create.

If the directory specified by `ODP_CONNECT_USER_DIR` already exists, the script exits without performing any action.

If intermediate directories in the path don't already exist, the script creates them. For example, if the user directory path is `/var/omdp/config`, but the `omdp` directory doesn't already exist, the script creates it.

The `create` action copies a sample `connect.yaml` configuration file from the installation directory as a starting point for you to edit.

The `create` action also copies another file, `connect.sample.yaml`, containing numerous examples of configuration parameters. OMEGAMON Data Connect does not use `connect.sample.yaml`. Use the examples in `connect.sample.yaml` as a reference for inserting new parameters into `connect.yaml`.

You can use the `create` action regardless of how you plan to run OMEGAMON Data Connect. For example, on z/OS, you can use the script to perform the `create` action, and then use JCL to run OMEGAMON Data Connect.

The `create` action assumes that the script is located in the OMEGAMON Data Connect installation directory. The `create` action uses the path of the running script to locate the sample configuration files under the installation directory.

`run` action

```
connect run
```

The `run` action uses a Java command line to start OMEGAMON Data Connect.

Before performing the `run` action, set the environment variable `ODP_CONNECT_USER_DIR` to the path of an existing user directory that you want to use to run OMEGAMON Data Connect.

The `run` action expects to find a configuration file at:

${ODP_CONNECT_USER_DIR}`/config/connect.yaml`

The `run` action assumes that the script is located in the OMEGAMON Data Connect installation directory. The `run` action uses the path of the running script to locate the core JAR file and mapping extension JAR files in the installation directory.

If the `JAVA_HOME` environment variable is not set, the script uses the following shell command to get the path of the **java** command:

```
command -v java
```

Usage

Typical usage involves three steps:

1. Perform the script `create` action.
2. Edit the `config/connect.yaml` configuration file in the user directory to meet your site-specific requirements.
3. Perform the script `run` action.

Example: Create an OMEGAMON Data Connect user directory

Suppose that:

1. You want to create the OMEGAMON Data Connect user directory `/var/omdp/prod-a`
2. The OMEGAMON Data Connect installation directory is `/usr/lpp/omdp/kay-110`

At a shell prompt, change to the directory `/usr/lpp/omdp/kay-110/bin`, and then enter the following command line:

```
ODP_CONNECT_USER_DIR=/var/omdp/prod-a ./connect create
```

- This example shows a single command line that sets the environment variable `ODP_CONNECT_USER_DIR` and then runs the script. Instead of setting the variable each time you run the script, consider using an **`export`** shell command to set the variable in your user profile.
- The dot and slash (`./`) preceding the script name `connect` assumes that the script is in the current directory. That is, the current directory is the `bin` directory under the OMEGAMON Data Connect installation directory.
- Replace `/var/omdp/prod-a` with your user directory path.

Example: Run OMEGAMON Data Connect

Suppose that:

1. You have performed the `create` action in the previous example.
2. You have edited the configuration file in `/var/omdp/prod-a/config/connect.yaml` to meet your site-specific requirements.

To run OMEGAMON Data Connect, enter the following shell command line:

```
ODP_CONNECT_USER_DIR=/var/omdp/prod-a ./connect run
```

Related reference

Java command line to run OMEGAMON Data Connect
Whichever platform you choose, you can use a Java command line to run OMEGAMON Data Connect.

Sample JCL procedure to run OMEGAMON Data Connect
You can use JCL to run OMEGAMON Data Connect on z/OS.

OMEGAMON Data Connect user directory
An OMEGAMON Data Connect user directory contains files that configure OMEGAMON Data Connect for your site.

Java command line to run OMEGAMON Data Connect

Whichever platform you choose, you can use a Java command line to run OMEGAMON Data Connect.

For example:

```
java \
  -Xms1024m -Xmx4096m -XX:+ExitOnOutOfMemoryError \                    1
  -Dfile.encoding=ISO8859-1 \                                          2
  -Dodp.ext=<odp_extensions_paths_list> \                              3
```

```
-jar <odp_installation_directory>/lib/odp-server.jar \                    4
--spring.config.additional-location=<odp_user_directory>/config/connect.yaml    5
```

1

The initial heap size (`-Xms`) and maximum heap size (`-Xmx`) shown here are examples only.

The maximum heap size must be large enough to accommodate the OMEGAMON Data Connect queue capacity of each output. In practice, the default queue capacity meets typical requirements and fits within this example maximum heap size.

For more information on setting the heap size, see the advice on how to do heap sizing in the IBM Semeru Runtime Certified Edition for z/OS documentation.

2

Setting the file encoding option avoids a potentially inappropriate system default encoding, such as EBCDIC on z/OS.

3

The `-Dodp.ext` runtime option specifies the value of the custom Java system property `odp.ext`.

`odp.ext` specifies the location of OMEGAMON Data Connect mapping extension JAR files. The value of `odp.ext` is a comma-separated list of directory paths and individual `.jar` file paths. In the directory paths, OMEGAMON Data Connect looks for files with the extension `.jar`. OMEGAMON Data Connect does not recurse into subdirectories.

Mapping extensions extend OMEGAMON Data Connect to support different types of incoming data. OMEGAMON Data Connect supplies mapping extension JAR files in the `lib/ext` directory under the installation directory. You can also choose to store mapping extension JAR files in the user directory.

The sample JCL procedure and shell script for running OMEGAMON Data Connect set a default value for `-Dodp.ext` that includes the `lib/ext` directory under the installation directory and the `extensions` directory under the user directory:

```
-Dodp.ext=<odp_installation_directory>/lib/ext,\
          <odp_user_directory>/extensions
```

If a mapping extension JAR file for an agent exists in more than one location, then OMEGAMON Data Connect uses the latest version of the file.

4

`odp-server.jar` is a symbolic link that refers to the OMEGAMON Data Connect core JAR file, `odp-server-`*`version`*`.jar`.

It is your choice whether to use the symbolic link, which has no version in its file name, or the original file with a version in its name. To avoid updating the command for each version, use the symbolic link.

5

`<odp_user_directory>``/config/connect.yaml` is the location of your site-specific OMEGAMON Data Connect configuration file. To avoid updates to the sample configuration file overwriting your site-specific changes, store your configuration file in a user directory separate from the OMEGAMON Data Connect installation directory.

Platform-specific options

In some cases, on some platforms, you might need to specify additional Java command-line options.

For example, on z/OS, if you use Transport Layer Security (TLS) with the store type JCERACFKS to specify a RACF key ring with the `safkeyring` protocol, then you need to ensure that the Java virtual machine (JVM) includes the IBMZSecurity provider to handle that protocol. You can include the IBMZSecurity provider by specifying the following command-line option:

```
-Djava.protocol.handler.pkgs=com.ibm.crypto.zsecurity.provider
```

Related reference

Sample shell script to run OMEGAMON Data Connect
You can use a shell script to run OMEGAMON Data Connect on operating systems such as z/OS UNIX and Linux. In addition to running OMEGAMON Data Connect, the supplied sample script can also create a user directory containing a sample OMEGAMON Data Connect configuration file as a starting point for you to edit.

Sample JCL procedure to run OMEGAMON Data Connect
You can use JCL to run OMEGAMON Data Connect on z/OS.

Event publisher parameters
OMEGAMON Data Connect event publisher parameters control aspects of internal OMEGAMON Data Connect processing.

Modifying running components of OMEGAMON Data Provider

To control components of OMEGAMON Data Provider that are running on z/OS , you can use MVS system commands, such as **MODIFY**, **STOP**, and **START**.

Reloading OMEGAMON collection configuration

After updating collection configuration parameters in the RKANPARU(KAYOPEN) member, you need to apply the configuration changes to the affected collection tasks. To apply the changes, you can either restart the jobs that run the collection tasks, or enter the MVS **MODIFY** system command presented here.

About this task

You only need to restart or modify the jobs that are affected by the changes to the KAYOPEN member. For example, if you only edited parameters that select the collections for product name kc5, then you only need to restart or modify the job that runs the CICS monitoring agent.

For monitoring agents that run in the monitoring server address space (TEMS), such as the z/OS and storage agents, you need to restart or modify the TEMS job (example job name: OMEGDS).

Procedure

Enter the following **MODIFY** command for each job:

```
F job_name,KPQ,RELOAD_CONFIG,KAY
```

What to do next

To confirm the configuration changes, read the KAYL0005I messages in the RKLVLOG output data set of the job.

Related tasks

Configuring OMEGAMON monitoring agents as a data source
To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you need to configure the runtime member RKANPARU(KAYOPEN) to specify which historical collections to send to OMEGAMON Data Broker.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

Reloading OMEGAMON Data Broker configuration

After updating OMEGAMON Data Broker configuration parameters in the Zowe cross-memory server configuration member, you need to apply those changes to OMEGAMON Data Broker. To apply the changes, you need to restart (stop and then start) the Zowe cross-memory server.

Procedure

1. Stop the Zowe cross-memory server started task.

For example, enter the following MVS **STOP** system command:

```
P KAYSIS01
```

2. Start the Zowe cross-memory server started task.
   For example, enter the following MVS **START** system command:

```
S KAYSIS01
```

What to do next

To confirm the configuration changes, check the expected messages from OMEGAMON Data Broker.

Related reference

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

Expected messages
These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Reloading OMEGAMON Data Connect configuration

After updating the contents of an OMEGAMON Data Connect user directory, such as the `connect.yaml` configuration file or mapping extension JAR files, you need to apply those changes to the instances of OMEGAMON Data Connect that refer to the user directory. To apply the changes, you need to restart (stop and then start) the affected instances of OMEGAMON Data Connect.

Procedure

Stop and then start each affected instance of OMEGAMON Data Connect.

Method used to run OMEGAMON Data Connect	How to stop and then start OMEGAMON Data Connect

z/OS started task	a. Enter an MVS **STOP** system command. For example:
	`P KAYCONN`
	b. Enter an MVS **START** system command. For example:
	`S KAYCONN`

Shell script	Follow your site-specific practices to stop and then start the shell script.

What to do next

To confirm the configuration changes, check the expected messages from OMEGAMON Data Connect.

Related reference

OMEGAMON Data Connect configuration parameters
OMEGAMON Data Connect configuration parameters identify inputs, such as the TCP port on which to listen for data from OMEGAMON Data Broker, and outputs, such as destination analytics platforms. You can filter which data to output.

Expected messages

These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Displaying OMEGAMON Data Broker status

You can enter MVS **MODIFY** system commands to display information about the status of OMEGAMON Data Broker.

Procedure

Issue one of the following MVS **MODIFY** system commands to the Zowe cross-memory server that runs OMEGAMON Data Broker:

- To display the status of connections to OMEGAMON Data Connect:

```
F KAYSIS01,D(KAYB) FWD
```

- To display the status of stores, such as how many records OMEGAMON Data Broker has sent to OMEGAMON Data Connect:

```
F KAYSIS01,D(KAYB) STORE
```

where:

- KAYSIS01 is the name of the Zowe cross-memory server job.
- The fixed value KAYB identifies the OMEGAMON Data Broker plug-in as the target of the command.

Changing OMEGAMON Data Broker network activity logging level

You can enter an MVS **MODIFY** system command to change the logging level of OMEGAMON Data Broker network activity dynamically, while OMEGAMON Data Broker is running.

About this task

In OMEGAMON Data Broker, each forwarder has a logging level. The logging level determines the level of network activity each forwarder logs. The default level is 0 (none).

Typically, you only need to set the logging level if IBM Software Support requests you to do so for troubleshooting. The logging level does not affect KAYB-prefix messages from OMEGAMON Data Broker.

You can set the logging level in the forwarder parameters in the configuration member of the Zowe cross-memory server that runs OMEGAMON Data Broker.

Alternatively, you can use the MVS **MODIFY** system command described here to set the logging level *dynamically*, overriding the logging level set in the configuration member.

Procedure

Issue an MVS **MODIFY** system command to the Zowe cross-memory server that runs OMEGAMON Data Broker:

```
F KAYSIS01,S(KAYB) FWD LOGOPTS(VERBOSITY) <forwarder_id> <log_level>
```

where:

- KAYSIS01 is the name of the Zowe cross-memory server job.
- The fixed value KAYB identifies the OMEGAMON Data Broker plug-in as the target of the command.
- *<forwarder_id>* matches the corresponding forwarder parameters in the configuration member.
- Allowed values for *<log_level>*:

<log_level>	**Description**	**Notes**
0	None	Default value
1	Fatal	Only log fatal errors
2	Error	Log all errors
3	Warning	Log any warnings
4	Info	Log informational messages
5	Verbose	Log verbose informational messages
6	Debug	Log messages useful for debugging
7	Trace	Log low-level trace messages
8	All	Log all messages

Higher logging levels include all messages from lower levels. For example, level 4 (info) includes all warnings and errors.

The details in messages at levels 6 and higher are intended for use only by IBM Software Support.

To reset the logging level to the value set in the configuration member or, if the member does not set a value, to 0, specify the keyword `DEFAULT` instead of a numeric level.

Results

OMEGAMON Data Broker logs network activity messages to the Zowe cross-memory server job step output data set with ddname SYSOUT (`//SYSOUT DD`) or, if the server job step does not define that ddname, then to data sets with the ddname SYS*<nnnnn>* (`SYS00001`, `SYS00002`, etc.).

Example

The following command sets the `OM` forwarder logging level to 7:

```
F KAYSIS01,S(KAYB) FWD LOGOPTS(VERBOSITY) OM 7
```

The following command resets the `OM` forwarder logging level to the value set in the configuration member or, if the member does not set a value, to 0:

```
F KAYSIS01,S(KAYB) FWD LOGOPTS(VERBOSITY) OM DEFAULT
```

Related reference

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

Adding more OMEGAMON collections to OMEGAMON Data Provider

If you have already configured an OMEGAMON runtime environment to send collections to OMEGAMON Data Provider, then follow the steps here to add more.

Procedure

1. Update the collection configuration in the RKANPARU(KAYOPEN) member to select the additional collections.

   Tip: Update KAYOPEN before you create the collections. Performing the configuration in this order ensures that attributes go to the correct destinations as soon as you create the collections.
2. Reload the collection configuration in the affected monitoring agents.
3. If necessary, update and then reload the OMEGAMON Data Connect configuration.

   Whether you need to perform this step depends on whether the current OMEGAMON Data Connect configuration already selects the corresponding table for forwarding. For example, if OMEGAMON Data Connect is already configured to forward *all* tables from a monitoring agent, and you are adding a collection for another table from that agent, then you don't need to perform this step.
4. Create the collections.

   To create historical collections, you use the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP). For more information about creating historical collections, see the OMEGAMON documentation for e3270UI and TEP.
5. Check the RKLVLOG output data set of the affected monitoring agent jobs for KPQH038I messages.
6. Check the STDOUT output file from OMEGAMON Data Connect for KAYC0008I and KAYC0033I messages.
7. Check that the attributes are arriving at your analytics platform.

   For example, in Elastic, check that an index has been created for the table.

Related tasks

Configuring OMEGAMON monitoring agents as a data source
To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you need to configure the runtime member RKANPARU(KAYOPEN) to specify which historical collections to send to OMEGAMON Data Broker.

Reloading OMEGAMON collection configuration
After updating collection configuration parameters in the RKANPARU(KAYOPEN) member, you need to apply the configuration changes to the affected collection tasks. To apply the changes, you can either restart the jobs that run the collection tasks, or enter the MVS **MODIFY** system command presented here.

Reloading OMEGAMON Data Connect configuration
After updating the contents of an OMEGAMON Data Connect user directory, such as the `connect.yaml` configuration file or mapping extension JAR files, you need to apply those changes to the instances of OMEGAMON Data Connect that refer to the user directory. To apply the changes, you need to restart (stop and then start) the affected instances of OMEGAMON Data Connect.

Related reference

OMEGAMON Data Connect configuration parameters
OMEGAMON Data Connect configuration parameters identify inputs, such as the TCP port on which to listen for data from OMEGAMON Data Broker, and outputs, such as destination analytics platforms. You can filter which data to output.

Related information

OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

Configuration

OMEGAMON Data Provider involves four configuration points: the data source, the two components of OMEGAMON Data Provider (OMEGAMON Data Broker and OMEGAMON Data Connect), and the destination.

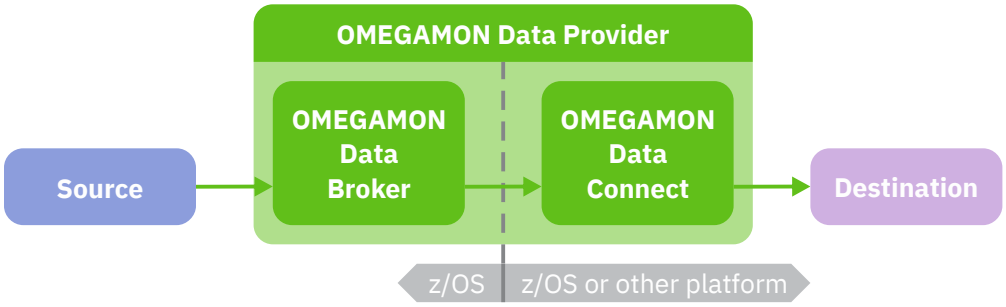


Figure 16. Configuration points: source, Broker, Connect, destination

This OMEGAMON Data Provider documentation includes details for the following configuration points:

- OMEGAMON monitoring agents as a data source
- OMEGAMON Data Broker
- OMEGAMON Data Connect

This OMEGAMON Data Provider documentation also includes *basic* configuration details for some destinations. See "Integrating analytics platforms with OMEGAMON Data Provider" on page 63.

For other data sources, and *comprehensive* details on configuring destinations, see the separate documentation for that software.

Configuration parameters for OMEGAMON monitoring agents as a data source

OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

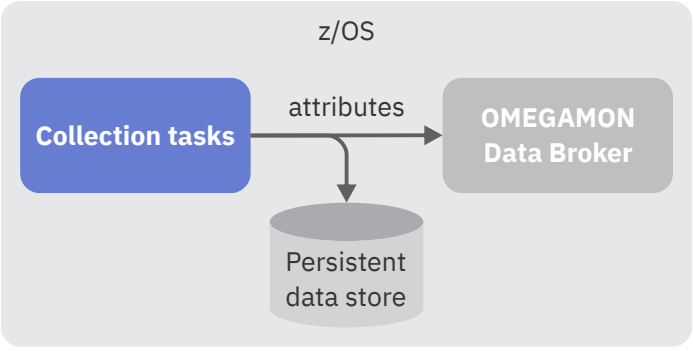


Figure 17. OMEGAMON Data Provider collection configuration parameters control where attributes are sent

Selecting versus creating collections

These parameters *select* collections; they do not *create* collections.

Historical collections are a prerequisite for using OMEGAMON Data Provider. Before configuring OMEGAMON Data Provider, you need to create historical collections.

To create historical collections, you use the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP). For more information about creating historical collections, see the OMEGAMON documentation for e3270UI and TEP.

Tip: You can specify these parameters to select collections *before* you create the corresponding collections. Configuring these parameters first means that, when you create the collections, collection tasks immediately send the attributes to the appropriate destinations.

Format

```
broker:
  name: <string>
collections:
  - product: k<pp> # Product code (example: km5)
    table: <table_name>
    interval: <minutes> # 0 matches any interval
    destination: # Either or both
      - pds
      - open
  - ... # More collections
```

The OMEGAMON Data Provider collection configuration member is a YAML document. The configuration parameters and their values conform to YAML syntax.

Tip: To check your configuration parameters, use the OMEGAMON Data Provider configuration validator.

Parameter names and values are case-insensitive, with one exception: the broker name is case-sensitive.

Character encoding

Collection tasks use EBCDIC code page 1047 to interpret the characters of the configuration member.

The code page is significant only if you use characters outside of the "invariant subset" of EBCDIC: characters that have different byte values in different EBCDIC code pages. For example, square brackets ([ ]) have different byte values in EBCDIC code pages 037 and 1047.

If you do use such characters, then when you edit the configuration member on z/OS, ensure that your terminal code page is set to EBCDIC code page 1047. For example, in your terminal emulator settings. Otherwise, you risk introducing byte values that your terminal displays as one character but that represents a different character when interpreted using EBCDIC code page 1047.

Tip: To avoid such code page issues, only use characters in the invariant subset of EBCDIC. In particular, do not use square brackets.

To avoid square brackets in YAML, use the block sequence YAML syntax shown in this documentation, not flow sequences. Block sequences are delimited by newlines and hyphens, whereas flow sequences are enclosed in square brackets.

Location

If you choose to specify this optional configuration member, then it must be member name KAY0PEN in the RKANPARU library of your OMEGAMON runtime environment (RTE).

If you omit this member, then OMEGAMON Data Provider is dormant and attributes from historical collections are sent to PDS only.

Attention: The KAY0PEN member is not managed by PARMGEN or Monitoring Configuration Manager. Some actions of PARMGEN and Monitoring Configuration Manager, such as the **GENERATE** action of Monitoring Configuration Manager, delete RKANPARU library members. PTF UJ93077 for APAR OA64681 (2Q23) excludes members with the name pattern KAY*, such as

KAYOPEN, from being deleted. If your site does not yet have that PTF applied, then you must maintain your primary copy of KAYOPEN in a different location of your choice and, after each GENERATE action, copy KAYOPEN to the RKANPARU library.

A sample member is supplied in TKANSAM(KAYOPEN).

Parameter descriptions

broker
Contains a single child key:

name
The name of the Zowe cross-memory server that runs the OMEGAMON Data Provider to which you want to send data.

This name is the value of the **NAME** runtime parameter of the JCL **EXEC** statement for the KAYSIS01 program (corresponding default procedure and job name: KAYSIS01).

Example value: ODP_BROKER

collections
Specifies a block sequence of historical collections. Each entry in the sequence is marked by a dash and space.

Each entry selects a historical collection that you have created in OMEGAMON and specifies destinations for that collection.

Each entry uses a combination of three values to select a historical collection: product code, table name, and collection interval.

To send data from a collection to OMEGAMON Data Broker, you must select the collection and specify the destination open.

product
The 3-character k*pp* product code of the monitoring agent that owns the table.

table
The table name. For example, ascpuutil (Address Space CPU Utilization).

interval
The collection interval in minutes or the special value 0 (zero).

The value 0 acts as a wildcard; it selects all historical collections for the table, regardless of collection interval.

Examples of minute values:

1
Every minute

5
Every 5 minutes

15
Every 15 minutes

30
Every 30 minutes

60
Every hour

1440
Once per day

To select a collection, either specify the wildcard value 0 or the number of minutes that matches the specific collection interval.

For example, to select a collection that has a collection interval of 1 day, specify `interval: 1440`.

Specifying `interval: 0` offers flexibility: it means that you can change the collection interval of a collection without having to specify that different `interval` value here and then restart or modify running OMEGAMON monitoring agents.

If you have multiple collections for the same table, but with different collection intervals, then you can choose to send them all to the same destinations with a single entry that specifies `interval: 0`, or you can specify multiple entries with specific collection intervals.

`destination`
Specifies a sequence of destinations for the table.

The sequence can contain either or both of the following values:

open
Send data from this collection to OMEGAMON Data Broker.

pds
Send data from this collection to the persistent data store.

If you want to view attributes from this collection in the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP) user interface, or store the attributes in Tivoli Data Warehouse, then you must include `pds` as a destination.

To pass attributes directly through to OMEGAMON Data Broker without storing them on disk (in the PDS), specify `open` as the only destination.

For an overview of the choice of destinations, see "OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider" on page 42.

You can specify destinations either in a block sequence, delimited by line breaks and hyphens:

```
destination:
  - open
  - pds
```

or in a flow sequence, delimited by commas and wrapped in square brackets:

```
destination: [open, pds]
```

Precedence of entries that select the same collections

If more than one entry in the `collections` sequence specifies the same combination of product name, table name, and collection interval, then the last entry takes precedence. That is, collections will be sent to the destinations specified by the last entry.

Entries with a specific interval value take precedence over entries with the wildcard interval value of 0.

Default destinations of unselected collections

The following conditions determine the default destination for collections that are not selected by any of the entries in the `collections` sequence:

Condition	Destination
No entries select that combination of product code and table name.	PDS only.
One or more entries select that combination of product code and table name, but none of those entries select that collection interval.	None.

Condition	Destination
	The collection is discarded. Data from that collection is not sent to either the PDS or OMEGAMON Data Broker.

Applying configuration changes

After editing this configuration member, you need to apply changes to the jobs that run the affected OMEGAMON monitoring agents.

You must either restart the jobs or enter an MVS **MODIFY** system command to reload their collection configuration.

Example: All collection intervals to both destinations

The following example selects collections for two tables; both tables are from the z/OS monitoring agent, product code km5.

```
broker:
  name: ODP_BROKER
collections:
  - product: km5
    table: ascpuutil
    interval: 0
    destination:
      - open
      - pds
  - product: km5
    table: km5msucap
    interval: 0
    destination:
      - open
      - pds
```

This example selects all collections for these tables, regardless of collection interval.

This example sends all selected collections to both the PDS and OMEGAMON Data Broker.

Collections for all other tables are sent to PDS only.

Example: Specific collection intervals

The following example only selects collections with the cited collection intervals.

```
broker:
  name: ODP_BROKER
collections:
  - product: km5
    table: ascpuutil
    interval: 1
    destination:
      - open
      - pds
  - product: km5
    table: km5msucap
    interval: 5
    destination:
      - open
      - pds
```

For table `ascpuutil`, this example only selects a collection that has a collection interval of 1 minute.

For table km5msucap, this example only selects a collection that has a collection interval of 5 minutes.

Collections for tables ascpuutil and km5msucap with other collection intervals are discarded.

Collections for all other tables are sent to PDS only.

Example: Multiple specific collection intervals

The following example sends collections for the same table, but with different collection intervals, to different destinations.

```
broker:
  name: ODP_BROKER
collections:
  - product: km5
    table: ascpuutil
    interval: 1
    destination:
      - open
  - product: km5
    table: ascpuutil
    interval: 5
    destination:
      - pds
```

A collection for table ascpuutil with a collection interval of 1 minute is sent to OMEGAMON Data Broker only.

A collection for table ascpuutil with a collection interval of 5 minutes is sent to the PDS only.

Collections for table ascpuutil with other collection intervals are discarded.

Collections for all other tables are sent to PDS only.

Example: All collection intervals to the PDS but only a specific collection interval to OMEGAMON Data Broker

Suppose that you want collections for a table to be sent to the PDS regardless of collection interval. However, you only want a collection for that table to be sent to OMEGAMON Data Broker if the collection has a specific collection interval.

The following example demonstrates using the special interval value 0 to select all collection intervals, and then using a separate entry for the same table to override that behavior for a specific collection interval.

```
broker:
  name: ODP_BROKER
collections:
  - product: kc5
    table: kcpplx
    interval: 0
    destination:
      - pds
  - product: kc5
    table: kcpplx
    interval: 1 # Specific value takes precedence over wildcard (0)
    destination:
      - pds
      - open
```

All collections for CICS (kc5) table kcpplx are sent to the PDS, regardless of collection interval.

A collection for that table with a collection interval of 1 minute is sent to OMEGAMON Data Broker.

Collections for all other tables are sent to PDS only.

Related concepts

How OMEGAMON Data Provider extends OMEGAMON attribute collection
OMEGAMON Data Provider extends OMEGAMON attribute collection by introducing a new runtime environment member, RKANPARU(KAYOPEN), that sets the destination of collected attributes: PDS, OMEGAMON Data Provider, both, or neither.

Prerequisites for OMEGAMON monitoring agents as a data source
These prerequisites apply only if you are using OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider.

Related tasks

Configuring OMEGAMON monitoring agents as a data source
To use OMEGAMON monitoring agents as a data source for OMEGAMON Data Provider, you need to configure the runtime member RKANPARU(KAYOPEN) to specify which historical collections to send to OMEGAMON Data Broker.

Reloading OMEGAMON collection configuration
After updating collection configuration parameters in the RKANPARU(KAYOPEN) member, you need to apply the configuration changes to the affected collection tasks. To apply the changes, you can either restart the jobs that run the collection tasks, or enter the MVS **MODIFY** system command presented here.

Related reference

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

OMEGAMON attribute dictionary
OMEGAMON Data Connect includes a dictionary of OMEGAMON attributes in a set of YAML files.

Related information

OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

OMEGAMON Data Broker configuration parameters

OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

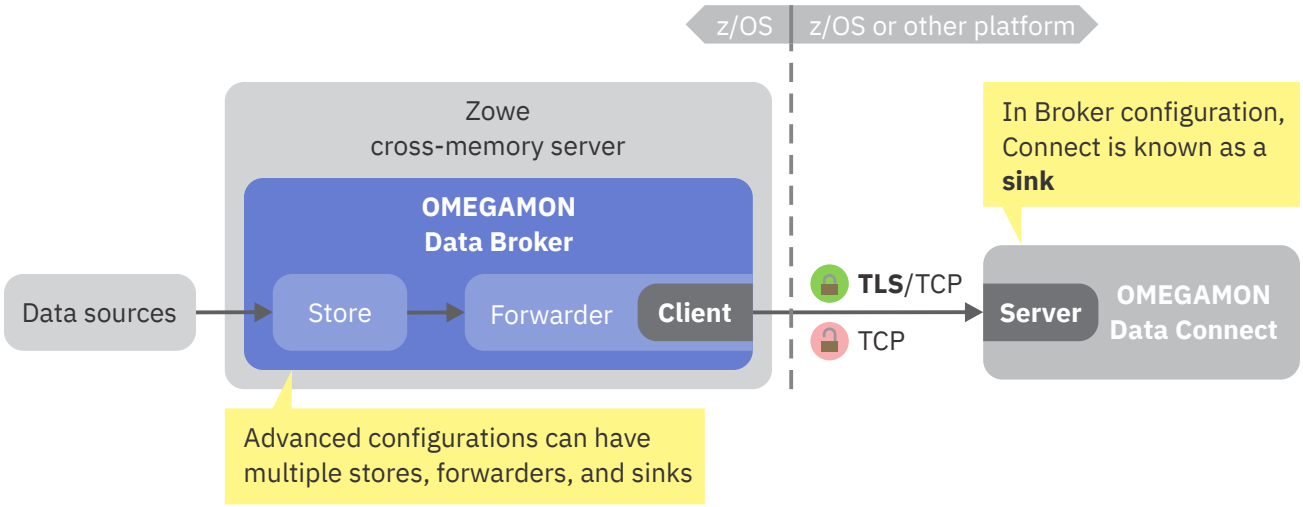


Figure 18. OMEGAMON Data Broker configuration points: store, forwarder, and sink

In the context of the TCP connection between OMEGAMON Data Broker and OMEGAMON Data Connect, OMEGAMON Data Broker is the *client* and OMEGAMON Data Connect is the *server*.

OMEGAMON Data Broker receives data from a data source into an internal store, and then forwards the data to OMEGAMON Data Connect.

To configure OMEGAMON Data Broker, you define a *forwarder* with a store as its *source* and an instance of OMEGAMON Data Connect as its *sink*.

A basic configuration has a single store, forwarder, and sink. Advanced configurations can have multiple stores, forwarders, and sinks.

An OMEGAMON Data Broker configuration consists of the following types of parameters:

General parameters
Register Zowe cross-memory server plug-ins, including OMEGAMON Data Broker.

Store parameters
Define the stores that data sources write to. When a data source sends data to OMEGAMON Data Broker, the data source specifies a store name. The OMEGAMON Data Broker configuration must include store parameters that define a store with that name. A data source can specify a store name that is used only by that data source, or a store name that is also used by other data sources.

Sink parameters
Define a connection to OMEGAMON Data Connect.

Forwarder parameters
Link stores to sinks. Each forwarder links one store to one sink.

Format

Only a few parameters are required; most are optional.

The following listing shows a minimal complete set of required parameters for a basic configuration:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001
KAY.CIDB.STORE.<store_id>.NAME=<store_name>
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.<forwarder_id>.SOURCE_STORE=<store_name>
KAY.CIDB.FWD.<forwarder_id>.SINK_HOST=<connect_hostname>
KAY.CIDB.FWD.<forwarder_id>.SINK_PORT=<connect_port>
```

The order of parameters in the configuration member is not significant. The order presented here is a typical order only.

OMEGAMON Data Broker configuration parameter names are case-sensitive.

The following listings show the comprehensive set of required and optional parameters.

General parameters:

```
* General parameters: required
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001
```

Store parameters:

```
* Store parameters: required
KAY.CIDB.STORE.<store_id>.NAME=<store_name>

* Store parameters: optional
KAY.CIDB.STORE.<store_id>.CELL.<cell_id>.SIZE=<bytes>
KAY.CIDB.STORE.<store_id>.CELL.<cell_id>.CAPACITY=<number>
KAY.CIDB.STORE.<store_id>.QUEUE.CAPACITY=<number>
```

Forwarder and sink parameters have two alternative formats:

Forwarder parameters with *forwarder-scope* sink parameters
Forwarder-scope parameters apply only to that forwarder:

```
* Forwarder parameters: required
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.<forwarder_id>.SOURCE_STORE=<store_name>
```

```
* Forwarder-scope sink parameters: required
KAY.CIDB.FWD.<forwarder_id>.SINK_HOST=<connect_hostname>
KAY.CIDB.FWD.<forwarder_id>.SINK_PORT=<connect_port>

* Forwarder parameters: optional
KAY.CIDB.FWD.<forwarder_id>.CONNECT_RETRY_INTERVAL=<seconds>
KAY.CIDB.FWD.<forwarder_id>.MAX_CONNECT_RETRY_ATTEMPTS=<number>
KAY.CIDB.FWD.<forwarder_id>.RECORD_QUEUE_LIMIT=<records>
KAY.CIDB.FWD.<forwarder_id>.MAX_BATCH_SIZE=<records>
KAY.CIDB.FWD.<forwarder_id>.LOGOPTS=--verbosity <log_level>

* Forwarder-scope sink parameters: optional
KAY.CIDB.FWD.<forwarder_id>.CONNECT_TIMEOUT=<seconds>
KAY.CIDB.FWD.<forwarder_id>.RECEIVE_TIMEOUT=<seconds>
KAY.CIDB.FWD.<forwarder_id>.SEND_TIMEOUT=<seconds>
KAY.CIDB.FWD.<forwarder_id>.PROTOCOL_VERSION=<protocol_version>

* Forwarder-scope sink SSL parameters: only required if sink connection
uses SSL/TLS
KAY.CIDB.FWD.<forwarder_id>.SECURITY=TLSv1.2
KAY.CIDB.FWD.<forwarder_id>.FIPS=<ON|OFF>
KAY.CIDB.FWD.<forwarder_id>.KEYRING=<string>
KAY.CIDB.FWD.<forwarder_id>.STASH=<string>
KAY.CIDB.FWD.<forwarder_id>.PASSWORD=<string>
KAY.CIDB.FWD.<forwarder_id>.CIPHERS=<string>
KAY.CIDB.FWD.<forwarder_id>.CERTLABEL=<string>

* Optional: More forwarders...
```

Forwarder parameters that refer to *reusable* sink parameters

Multiple forwarders can refer to the same set of reusable sink parameters:

```
* Reusable sink parameters: required
KAY.CIDB.SINK.<sink_id>.HOST=<connect_hostname>
KAY.CIDB.SINK.<sink_id>.PORT=<connect_port>

* Reusable sink parameters: optional
KAY.CIDB.SINK.<sink_id>.CONNECT_TIMEOUT=<seconds>
KAY.CIDB.SINK.<sink_id>.RECEIVE_TIMEOUT=<seconds>
KAY.CIDB.SINK.<sink_id>.SEND_TIMEOUT=<seconds>
KAY.CIDB.SINK.<sink_id>.PROTOCOL_VERSION=<protocol_version>

* Reusable sink SSL parameters: only required if sink connection uses
SSL/TLS
KAY.CIDB.SINK.<sink_id>.SECURITY=TLSv1.2
KAY.CIDB.SINK.<sink_id>.FIPS=<ON|OFF>
KAY.CIDB.SINK.<sink_id>.KEYRING=<string>
KAY.CIDB.SINK.<sink_id>.STASH=<string>
KAY.CIDB.SINK.<sink_id>.PASSWORD=<string>
KAY.CIDB.SINK.<sink_id>.CIPHERS=<string>
KAY.CIDB.SINK.<sink_id>.CERTLABEL=<string>

* Optional: More sinks...

* Forwarder parameters: required
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.<forwarder_id>.SOURCE_STORE=<store_name>

* Reference to reusable sink parameters: required
KAY.CIDB.FWD.<forwarder_id>.SINK=<sink_id>

* Forwarder parameters: optional
KAY.CIDB.FWD.<forwarder_id>.CONNECT_RETRY_INTERVAL=<seconds>
KAY.CIDB.FWD.<forwarder_id>.MAX_CONNECT_RETRY_ATTEMPTS=<number>
KAY.CIDB.FWD.<forwarder_id>.RECORD_QUEUE_LIMIT=<records>
```

```
KAY.CIDB.FWD.<forwarder_id>.MAX_BATCH_SIZE=<records>
KAY.CIDB.FWD.<forwarder_id>.LOGOPTS=--verbosity <log_level>

* Optional: More forwarders...
```

Reusable sink parameters are useful when you want multiple forwarders to use the same sink: you don't need to duplicate the sink parameters for each forwarder.

Forwarder-scope sink parameters take precedence over reusable sink parameters.

Tip: You can use forwarder-scope sink parameters to complete or individually override reusable sink parameters. For details, see "Example: Completing or overriding reusable sink parameters with forwarder-scope sink parameters" on page 111.

Location

OMEGAMON Data Broker configuration parameters are stored in the configuration member of the Zowe cross-memory server.

The configuration member name matches the following pattern:

*pppp*IP*xx*

where:

pppp
The first four characters of the Zowe cross-memory server load module name. OMEGAMON Data Provider supplies the Zowe cross-memory server load module in TKANMODP(KAYSIS01). The same load module is supplied with Zowe as ZWESIS01.

xx
The value of the optional **MEM** runtime parameter in the startup JCL for the Zowe cross-memory server. Default: 00.

For example:

Zowe cross-memory server load module name	MEM parameter in JCL	Configuration member name
KAYSIS01 (as supplied with OMEGAMON Data Provider)	00 (default)	KAYSIP00
KAYSIS01	01	KAYSIP01
ZWESIS01 (as supplied with Zowe)	00	ZWESIP00
ZWESIS01	01	ZWESIP01

The configuration member must be in a PARMLIB data set. Either:

- The data set specified by the PARMLIB ddname of the job step that runs the Zowe cross-memory server program.
- If that job step does not specify a PARMLIB ddname, the system PARMLIB. For example, SYS1.PARMLIB.

For example, given the following JCL after any symbol substitution:

```
//BROKER    EXEC PGM=KAYSIS01,REGION=0M,
//            PARM='NAME=ODP_BROKER,MEM=02'
//PARMLIB   DD DSNAME=MY.ODP.PARMLIB
```

the fully qualified dsname of the configuration member is MY.ODP.PARMLIB(KAYSIP02).

A sample member is supplied in TKANSAM(KAYSIP00).

Parameter namespaces and IDs

OMEGAMON Data Broker configuration parameters are namespaced. Each parameter name is prefixed by a sequence of period-delimited qualifiers that specify the context of the parameter.

For example, in the following parameter name:

```
KAY.CIDB.FWD.<forwarder_id>.SINK_HOST
```

KAY
Specifies that the parameter belongs to OMEGAMON Data Provider.

CIDB
Specifies that the parameter belongs to the OMEGAMON Data Broker component of OMEGAMON Data Provider.

FWD
Specifies that the parameter belongs to a forwarder.

<forwarder_id>
Specifies which forwarder the parameter belongs to.

A parameter namespace can include one or more IDs, such as *<store_id>*, *<sink_id>*, or *<forwarder_id>*.

An ID specifies an instance of an object and groups the parameters for that object. The qualifier preceding the ID specifies the object type, such as store (STORE), sink (SINK), or forwarder (FWD). Objects of the same type must use different IDs.

An ID is a case-sensitive string of 1 - 8 alphanumeric characters (a - z, A - Z, 0 - 9).

Example IDs: STORE1, SINK2, FWD3, OM, 1, 2, 3, A, B, C.

Note: A forwarder and a store can use the same ID, such as OM, but this does not imply any relationship between them. Each forwarder specifies the *name* of the store to use as its source.

No other Zowe cross-memory server configuration parameters required

If you're only using the Zowe cross-memory server to host OMEGAMON Data Broker, then the Zowe cross-memory server configuration member only needs to contain the following parameters:

- ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL, to register the ZIS dynamic linkage base plug-in that is required by OMEGAMON Data Broker.
- ZWES.PLUGIN.KAY.CIDB=KAYB0001, to register the OMEGAMON Data Broker plug-in.
- KAY.CIDB-namespace OMEGAMON Data Broker configuration parameters described here.

You don't need to specify any other ZWES-namespace parameters for the Zowe cross-memory server itself.

Splitting long parameter values over multiple lines

Some parameters can have long values. However, the Zowe cross-memory server and OMEGAMON Data Broker use only the first 71 characters of each record in the configuration member.

To split long parameter values over multiple lines, use a backslash (\) as a line continuation character. Example:

```
KAY.CIDB.FWD.FWD1.KEYRING=\
  /u/my/long/directory/path/to/\
  a-long-file-name.p12
```

Leading spaces on continuation lines are ignored.

General parameters

Required general parameters:

ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
Registers ZISDYNAMIC, the ZIS dynamic linkage base plug-in.

OMEGAMON Data Broker requires the ZISDYNAMIC plug-in.

OMEGAMON Data Provider supplies the ZISDYNAMIC plug-in load module in `TKANMODP(KAYSISDL)`. The same load module is supplied with Zowe as `ZWESISDL`.

If you are running OMEGAMON Data Broker in a Zowe cross-memory server in a separate installation of Zowe instead of the server supplied with OMEGAMON Data Provider, then use the following parameter name, without the KAY qualifier, and refer to the plug-in load module name as supplied with Zowe:

```
ZWES.PLUGIN.ZISDYNAMIC=ZWESISDL
```

ZWES.PLUGIN.KAY.CIDB=KAYB0001
Registers OMEGAMON Data Broker as a plug-in of the Zowe cross-memory server.

OMEGAMON Data Provider supplies the OMEGAMON Data Broker plug-in load module in `TKANMODP(KAYB0001)`.

These plug-in load modules must be available to the Zowe cross-memory server. To be available to the server, the load modules must be members of a data set that is specified by the `STEPLIB` ddname of the job step that runs the server program, `KAYSIS01`.

Store parameters

Required store parameter:

KAY.CIDB.STORE.*<store_id>*.NAME=*<store_name>*
Defines a store.

The store name depends on the data source. When a data source sends data to OMEGAMON Data Broker, the data source specifies a store name.

You must define stores to match the store names used by your data sources.

If you use OMEGAMON monitoring agents as a data source, you must define a store named `OMEGAMON`.

For store names used by other data sources, see the separate documentation for that software.

Different data sources can use the same store name.

Optional store parameters:

Tip: The optional store parameters expose details that users typically don't need to know. The default values work for typical data sources. These parameters are documented here only in case a data source requires values that are different to the defaults. Even in that case, you don't need to understand these parameters in enough detail to determine their values: the documentation for the data source will supply parameter values for you to copy into your configuration member.

KAY.CIDB.STORE.*<store_id>*.CELL.*<cell_id>*.SIZE=*<bytes>*
Cell size, in bytes.

A store consists of cells. OMEGAMON Data Broker puts each incoming record into a cell in the named store. Cells can be various sizes, to accommodate different sizes of incoming records. Record sizes depend on the data source.

The *<cell_id>* qualifier groups the parameters for this cell size.

KAY.CIDB.STORE.*<store_id>*.CELL.*<cell_id>*.CAPACITY=*<number>*
The initial number of cells of this *<cell_id>*; this size.

OMEGAMON Data Broker uses this value to preallocate memory for cells. During processing, OMEGAMON Data Broker allocates additional memory as required.

Store parameters define cells as pairs of SIZE and CAPACITY.

The default parameter values for cells specify various sizes and capacities. For example (actual default values do not necessarily exactly match this example):

```
KAY.CIDB.STORE.STORE1.CELL.1.SIZE=128
KAY.CIDB.STORE.STORE1.CELL.1.CAPACITY=1000

KAY.CIDB.STORE.STORE1.CELL.2.SIZE=256
KAY.CIDB.STORE.STORE1.CELL.2.CAPACITY=5000

KAY.CIDB.STORE.STORE1.CELL.3.SIZE=512
KAY.CIDB.STORE.STORE1.CELL.3.CAPACITY=5000

KAY.CIDB.STORE.STORE1.CELL.4.SIZE=1024
KAY.CIDB.STORE.STORE1.CELL.4.CAPACITY=1000

KAY.CIDB.STORE.STORE1.CELL.5.SIZE=2048
KAY.CIDB.STORE.STORE1.CELL.5.CAPACITY=1000

KAY.CIDB.STORE.STORE1.CELL.6.SIZE=4096
KAY.CIDB.STORE.STORE1.CELL.6.CAPACITY=1000

KAY.CIDB.STORE.STORE1.CELL.7.SIZE=8192
KAY.CIDB.STORE.STORE1.CELL.7.CAPACITY=200
```

You cannot override default values individually for each cell ID. Specifying *any* cell parameters for a store overrides the default values for *all* cell IDs for that store. If you specify any cell parameters, then you must specify a complete set.

KAY.CIDB.STORE.<*store_id*>.QUEUE.CAPACITY=<*number*>
The initial number of cells that the store's queues can contain. Default: 10000 (ten thousand).

A store can have multiple forwarders. Each forwarder has its own queue. This capacity is shared across all of the store's queues.

OMEGAMON Data Connect expands this capacity as required. However, while the total shared capacity can expand, each forwarder's queue has a maximum number of records, set by the forwarder's RECORD_QUEUE_LIMIT parameter.

Sink parameters

The namespace of a sink parameter determines its context: forwarder-scope or reusable.

Context	Namespace
Forwarder-scope	KAY.CIDB.FWD.<*forwarder_id*>
Reusable	KAY.CIDB.SINK.<*sink_id*>

Note: Forwarder-scope sink parameters and reusable sink parameters have the same names ("trailing qualifiers"), with two exceptions: SINK_HOST and SINK_PORT in forwarder-scope sink parameters versus HOST and PORT, without the SINK_ prefix, in reusable sink parameters.

Required forwarder-scope sink parameters:

KAY.CIDB.FWD.<*forwarder_id*>.SINK_HOST=<*connect_hostname*>
Hostname or IP address of the OMEGAMON Data Connect instance that is listening for data from OMEGAMON Data Broker.

In the context of the OMEGAMON Data Broker forwarder, OMEGAMON Data Connect is a *sink*: a destination.

If you plan to run OMEGAMON Data Connect on the same z/OS instance as OMEGAMON Data Broker, then you can specify the value `localhost` or the local loopback IP address. The typical local loopback IPv4 address is `127.0.0.1`.

KAY.CIDB.FWD.*<forwarder_id>*.SINK_PORT=*<connect_port>*
The port on which OMEGAMON Data Connect is listening. Follow your site-specific standards for assigning port numbers.

Required reusable sink parameters:

KAY.CIDB.SINK.*<sink_id>*.HOST=*<connect_hostname>*
See the previous description of the corresponding forwarder-scope parameter, SINK_HOST.

KAY.CIDB.SINK.*<sink_id>*.PORT=*<connect_port>*
See the previous description of the corresponding forwarder-scope parameter, SINK_PORT.

Optional forwarder-scope or reusable sink parameters:

<forwarder_or_sink_namespace>*.CONNECT_TIMEOUT=*<seconds>
Time in seconds to wait to establish a connection to OMEGAMON Data Connect. Default: 5.

<forwarder_or_sink_namespace>*.RECEIVE_TIMEOUT=*<seconds>
Receive timeout in seconds. Default: 300 (5 minutes).

<forwarder_or_sink_namespace>*.SEND_TIMEOUT=*<seconds>
Send timeout in seconds. Default: 0 (indefinite).

<forwarder_or_sink_namespace>*.PROTOCOL_VERSION=*<protocol_version>
The protocol version that OMEGAMON Data Broker uses to communicate with OMEGAMON Data Connect:

2
Version 2 (default). Sends a batch of records in each message, and waits for acknowledgment from OMEGAMON Data Connect before sending the next message.

Attention: If PROTOCOL_VERSION is set to 2, but the instance of OMEGAMON Data Connect does not support protocol version 2, then OMEGAMON Data Broker does not send data to that instance of OMEGAMON Data Connect.

1
Version 1. Sends only a single record in each message, and does not wait for acknowledgment from OMEGAMON Data Connect before sending the next message.

AUTO
Use the latest protocol version supported by OMEGAMON Data Connect.

Where possible, set a specific protocol version number instead of AUTO. In some cases, AUTO can inaccurately choose protocol version 1 when OMEGAMON Data Connect supports version 2. For example, if a proxy or load balancer is between OMEGAMON Data Broker and OMEGAMON Data Connect.

Sink SSL parameters

Sink SSL parameters are relevant only if you use TLS to secure the connection between OMEGAMON Data Broker and OMEGAMON Data Connect.

<forwarder_or_sink_namespace>*.SECURITY=*<string>
Enabled security protocols. Allowed values: TLSv1.2 or blank (no value). Default: no value; no security protocol.

Tip: For a connection without TLS, omit or comment-out the SECURITY parameter.

***<forwarder_or_sink_namespace>*.FIPS=ON|OFF**
Sets z/OS System SSL Federal Information Processing Standards (FIPS) mode. Default: OFF. For information about FIPS mode, see the z/OS System SSL documentation for your version of z/OS. For example, FIPS 140-2 support in z/OS 2.5.0.

<forwarder_or_sink_namespace>*.KEYRING=*<string>
Identifies the collection of security certificates required for this connection. Can be one of the following values:

SAF key ring
Specified in the format *<owner_user_id>*/*<key_ring_name>*. For example:

```
my/kay_keyring
```

If the current user owns the key ring, the current user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class. If another user owns the key ring, the current user must have UPDATE access to that resource.

Certificate private keys are not available when using a SAF key ring owned by another user, except for SITE certificates where CONTROL authority is given to IRR.DIGTCERT.GENCERT in the FACILITY class or for user certificates where READ or UPDATE authority is given to *<ring_owner>*.*<ring_name>*.LST resource in the RDATALIB class.

Key database
A key database created by the z/OS gskkyman utility. The key database is specified as a z/OS UNIX file path. For example:

```
/u/my/security/certs/kay.kdb
```

PKCS #12 file
Specified as a z/OS UNIX file path. For example:

```
/u/my/security/certs/kay.p12
```

PKCS #11 token
Specified in the format *TOKEN*/*<token_name>*. For example:

```
*TOKEN*/kay.pkcs11.token
```

The *TOKEN* qualifier indicates that the value refers to a PKCS #11 token rather than a SAF key ring.

If you specify a key database or PKCS #12 file, but you do not specify either a **STASH** parameter or a **PASSWORD** parameter, then OMEGAMON Data Broker looks for a stash file in the same directory as the key database or PKCS #12 file, and with the same base file name, but with `.sth` extension. For example, if the **KEYRING** parameter specifies the following z/OS UNIX file path:

```
/u/my/security/certs/kay.kdb
```

or:

```
/u/my/security/certs/kay
```

(with no extension)

then OMEGAMON Data Broker looks for a stash file at the following path:

```
/u/my/security/certs/kay.sth
```

<forwarder_or_sink_namespace>*.STASH=*<path>
z/OS UNIX file path of the stash file that contains the password for the key database or PKCS #12 file.

If **PASSWORD** is specified, **STASH** is ignored.

<forwarder_or_sink_namespace>*.PASSWORD=*<string>
Password for the key database or PKCS #12 file.

If **KEYRING** specifies a SAF key ring or PKCS #11 token, **PASSWORD** is ignored.

<forwarder_or_sink_namespace>*.CIPHERS=*<hex_string>
List of candidate cipher suites to try, in order. The list is a concatenation of 4-digit hexadecimal cipher suite numbers supported by z/OS System SSL. For example:

```
000A000D001000130016
```

If you omit **CIPHERS**, OMEGAMON Data Broker uses the system default list of cipher suites. That list depends on whether FIPS mode is on.

Tip: To match a z/OS System SSL cipher suite number to the corresponding OpenSSL cipher suite name, go to the z/OS System SSL documentation and look up the "short name" for that cipher suite in the table of cipher suite definitions. The short name is the name that is defined in the associated Request for Comments (RFC) by the Internet Engineering Task Force (IETF). Then go to the OpenSSL documentation for the **ciphers** command, and use the RFC name to find the corresponding OpenSSL name.

For more information on cipher suite definitions, see the z/OS System SSL documentation for your version of z/OS. For example, the cipher suite definitions supported by z/OS 2.5.0.

<forwarder_or_sink_namespace>*.CERTLABEL=*<string>
Specifies the label (also known as *alias*) of the client certificate that is used to authenticate OMEGAMON Data Broker (the client) to OMEGAMON Data Connect (server). The client certificate, and its private key, must be in the collection that is specified by the **KEYRING** parameter.

Default: no label.

CERTLABEL is only used if OMEGAMON Data Connect requires client authentication.

If OMEGAMON Data Connect requires client authentication, but you omit **CERTLABEL**, then OMEGAMON Data Broker uses the default certificate from the collection that is specified by the **KEYRING** parameter.

Forwarder parameters

You can define one or more forwarders. Use a different *<forwarder_id>* to group the parameters for each forwarder.

Required forwarder parameters:

KAY.CIDB.FWD=ON
Enables the forwarder subsystem of OMEGAMON Data Broker. The forwarder enables OMEGAMON Data Broker to send data over a TCP/IP network to a "sink" (forwarding destination) such as OMEGAMON Data Connect.

Values:

ON
Enables the forwarder. This value is case-sensitive.

If you omit this parameter, or specify any value other than ON in all uppercase, then the forwarder subsystem is disabled, and OMEGAMON Data Broker will not forward data.

You only need to specify this parameter once in the configuration member, not once per forwarder.

KAY.CIDB.FWD.*<forwarder_id>*.SOURCE_STORE=*<store_name>*
The name of an OMEGAMON Data Broker store to which a data source sends data.

The value of this parameter must match the value of a NAME store parameter.

The forwarder sends data from this store to the forwarder sink.

For OMEGAMON Data Provider, you must specify the store name OMEGAMON.

Optional forwarder parameters:

KAY.CIDB.FWD.*<forwarder_id>*.SINK=*<sink_id>*
Refers to a set of reusable sink parameters.

KAY.CIDB.FWD.*<forwarder_id>*.CONNECT_RETRY_INTERVAL=*<seconds>*
Number of seconds to wait before retrying connection to OMEGAMON Data Connect. Default: 20.

KAY.CIDB.FWD.*<forwarder_id>*.MAX_CONNECT_RETRY_ATTEMPTS=*<number>*
Maximum number of attempts to retry connection to OMEGAMON Data Connect. Default: no value; unlimited.

KAY.CIDB.FWD.*<forwarder_id>*.RECORD_QUEUE_LIMIT=*<records>*
The maximum number of records allowed in this forwarder's queue. The default value is `1000000` (one million) records.

When a queue is full, each new incoming record overwrites the oldest remaining record in the queue.

If OMEGAMON Data Connect is unavailable, then older records in the queue might be lost, overwritten by new records before they can be forwarded.

KAY.CIDB.FWD.*<forwarder_id>*.MAX_BATCH_SIZE=*<records>*
Maximum number of records that OMEGAMON Data Broker sends in a single message to OMEGAMON Data Connect. Must be an integer between 1 and `1000000` (one million). Default: `1000` (one thousand).

Does not apply to protocol version 1: see the sink parameter `PROTOCOL_VERSION`.

KAY.CIDB.FWD.*<forwarder_id>*.LOGOPTS=--verbosity *<log_level>*
The logging level of network activity for this forwarder. The default level is 0 (none).

Typically, you only need to set the logging level if IBM Software Support requests you to do so for troubleshooting. The logging level does not affect KAYB-prefix messages from OMEGAMON Data Broker.

Allowed values:

<log_level>	**Description**	**Notes**
0	None	Default value
1	Fatal	Only log fatal errors
2	Error	Log all errors
3	Warning	Log any warnings
4	Info	Log informational messages
5	Verbose	Log verbose informational messages
6	Debug	Log messages useful for debugging
7	Trace	Log low-level trace messages
8	All	Log all messages

Higher logging levels include all messages from lower levels. For example, level 4 (info) includes all warnings and errors.

The details in messages at levels 6 and higher are intended for use only by IBM Software Support.

OMEGAMON Data Broker logs network activity messages to the Zowe cross-memory server job step output data set with ddname SYSOUT (`//SYSOUT DD`) or, if the server job step does not define that ddname, then to data sets with the ddname SYS*<nnnnn>* (`SYS00001`, `SYS00002`, etc.).

Example parameter:

```
KAY.CIDB.FWD.FWD1.LOGOPTS=--verbosity 7
```

Tip: You can override the value of this parameter dynamically, while OMEGAMON Data Broker is running. For details, see “Changing OMEGAMON Data Broker network activity logging level” on page 85.

Related concepts
OMEGAMON Data Provider topology

OMEGAMON Data Provider topology typically consists of one instance of OMEGAMON Data Broker per z/OS LPAR, with multiple instances of OMEGAMON Data Broker feeding a single instance of OMEGAMON Data Connect.

Related tasks

Configuring OMEGAMON Data Broker
OMEGAMON Data Broker is a plug-in for the Zowe cross-memory server. You can configure OMEGAMON Data Broker using either the server that is supplied with OMEGAMON Data Provider (the best and easiest choice for most users) or a server in an existing separate Zowe installation.

Reloading OMEGAMON Data Broker configuration
After updating OMEGAMON Data Broker configuration parameters in the Zowe cross-memory server configuration member, you need to apply those changes to OMEGAMON Data Broker. To apply the changes, you need to restart (stop and then start) the Zowe cross-memory server.

Changing OMEGAMON Data Broker network activity logging level
You can enter an MVS **MODIFY** system command to change the logging level of OMEGAMON Data Broker network activity dynamically, while OMEGAMON Data Broker is running.

Related reference

TCP input parameters
OMEGAMON Data Connect TCP input parameters specify how OMEGAMON Data Connect listens for data over a TCP network from OMEGAMON Data Broker.

Basic OMEGAMON Data Broker configuration examples

Basic OMEGAMON Data Broker configurations consist of one store, one forwarder, and one sink (one instance of OMEGAMON Data Connect).

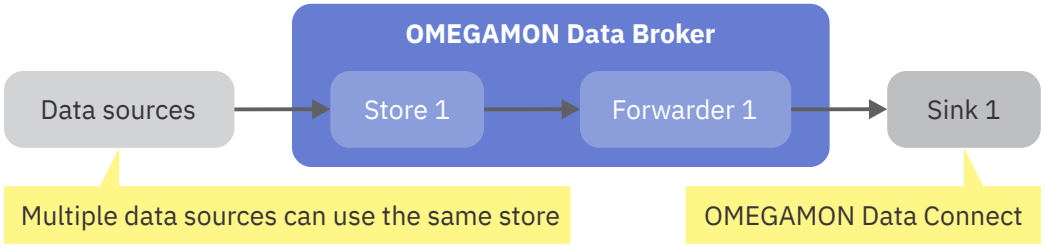


Figure 19. OMEGAMON Data Broker basic configuration: one store, one forwarder, one sink

Example: Forwarding without TLS

The following example configures OMEGAMON Data Broker to forward data to OMEGAMON Data Connect on the same z/OS instance as OMEGAMON Data Broker (`localhost`) and listening on port `15351`:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Forwarder and sink
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK_HOST=localhost
KAY.CIDB.FWD.FWD1.SINK_PORT=15351
```

A similar set of OMEGAMON Data Broker configuration parameters is supplied in the sample member `TKANSAM(KAYSIP00)`.

The following example shows the same configuration, but using *reusable* sink parameters instead of *forwarder-scope* sink parameters:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Sink
KAY.CIDB.SINK.SINK1.HOST=localhost
KAY.CIDB.SINK.SINK1.PORT=15351

* Forwarder
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1
```

Reusable sink parameters are especially useful in advanced configurations where multiple forwarders use the same sink. However, even in a basic configuration, you can choose to use reusable sink parameters to separate the different types of parameters.

Example: Forwarding with TLS using a RACF key ring

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Forwarder and sink
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK_HOST=localhost
KAY.CIDB.FWD.FWD1.SINK_PORT=15351
KAY.CIDB.FWD.FWD1.SECURITY=TLSv1.2
KAY.CIDB.FWD.FWD1.FIPS=ON
KAY.CIDB.FWD.FWD1.KEYRING=KAYSIS01/KAYring
```

This example is based on the following assumptions:

- You have configured the TCP input of OMEGAMON Data Connect to use TLSv1.2.
- At least one of the FIPS cipher suites specified here by OMEGAMON Data Broker matches a cipher suite specified by OMEGAMON Data Connect.
- You have created a RACF key ring named `KAYring`, owned by user `KAYSIS01` (the user that runs the Zowe cross-memory server instance that hosts the OMEGAMON Data Broker plug-in).
- The key ring contains a certificate that OMEGAMON Data Broker (the client) can use to authenticate OMEGAMON Data Connect (the server).
- OMEGAMON Data Connect does not require client authentication.

If OMEGAMON Data Connect requires client authentication, add the following parameter:

```
KAY.CIDB.FWD.FWD1.CERTLABEL=OMDPcert
```

where `OMDPcert` is the label (alias) of the client certificate in the key ring.

Here is the same configuration with reusable sink parameters:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001
```

```
* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Sink
KAY.CIDB.SINK.SINK1.HOST=localhost
KAY.CIDB.SINK.SINK1.PORT=15351
KAY.CIDB.SINK.SINK1.SECURITY=TLSv1.2
KAY.CIDB.SINK.SINK1.FIPS=ON
KAY.CIDB.SINK.SINK1.KEYRING=KAYSIS01/KAYring
KAY.CIDB.SINK.SINK1.CERTLABEL=OMDPcert

* Forwarder
KAY.CIDB.FWD=ON
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1
```

Advanced OMEGAMON Data Broker configuration examples

Advanced OMEGAMON Data Broker configurations consist of combinations of single or multiple stores, sinks, and forwarders.

The examples presented here show archetypal configuration patterns in separate instances of OMEGAMON Data Broker. Actual configurations can be more complex than these examples, combining multiple basic and advanced configuration patterns in a single instance of OMEGAMON Data Broker.

These examples use the term "multiple" to mean "two *or more*". For example, where an example shows two instances of a store, sink, or forwarder, you can specify more. The examples show only two instances for conciseness.

For store, sink, and forwarder IDs, use a naming convention of your choice. These examples use STORE*n*, SINK*n*, and FWD*n*. You can either use this generic naming convention or your own naming convention that is specific to your environment.

For consistency, modularity, and ease of maintenance, consider using *reusable* sink parameters rather than *forwarder-scope* sink parameters for all sinks, even if none of the forwarders use the same sink.

Example: Forwarding the same data to multiple instances of OMEGAMON Data Connect

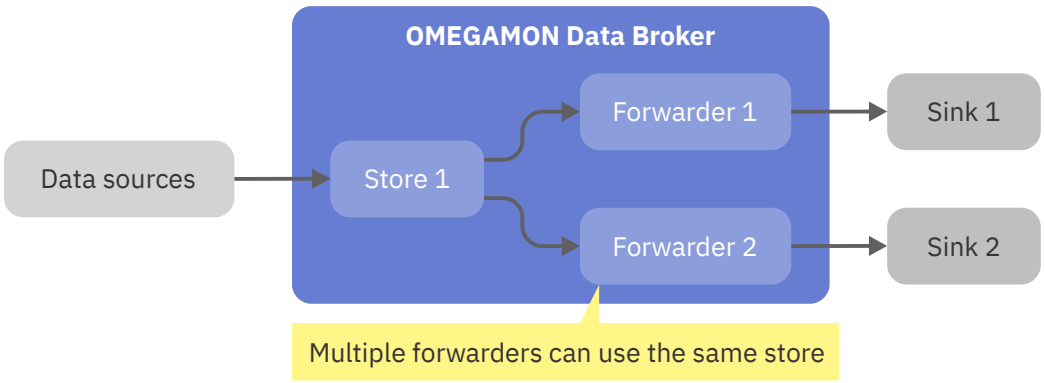


Figure 20. OMEGAMON Data Broker advanced configuration: one store, two forwarders, two sinks

There are various reasons why you might want to forward the same data to multiple instances of OMEGAMON Data Connect. For example:

- Different teams in your organization might want to independently manage their own instances of OMEGAMON Data Connect without affecting, or being affected by, other teams. Each team can change the configuration of OMEGAMON Data Connect according to their own requirements and restart OMEGAMON Data Connect according to their own schedule.
- If your organization uses OMEGAMON Data Provider to make data available to multiple destination analytics platforms, then, instead of configuring a single instance of OMEGAMON Data Connect that

sends data to all destinations, you might decide to use a separate instance of OMEGAMON Data Connect for each destination. Similar to the previous point about a team-based approach, this destination-based approach enables your organization to change the configuration of OMEGAMON Data Connect and restart OMEGAMON Data Connect according to the requirements of each analytics platform, without affecting others.

The following example configures OMEGAMON Data Broker to send data to two instances of OMEGAMON Data Connect, both running on the same z/OS instance as OMEGAMON Data Broker (`localhost`), but listening on different ports:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Sinks
KAY.CIDB.SINK.SINK1.HOST=localhost
KAY.CIDB.SINK.SINK1.PORT=15351

KAY.CIDB.SINK.SINK2.HOST=localhost
KAY.CIDB.SINK.SINK2.PORT=15352

* Forwarders
KAY.CIDB.FWD=ON

KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1

KAY.CIDB.FWD.FWD2.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD2.SINK=SINK2
```

Example: Forwarding data from multiple stores to the same instance of OMEGAMON Data Connect

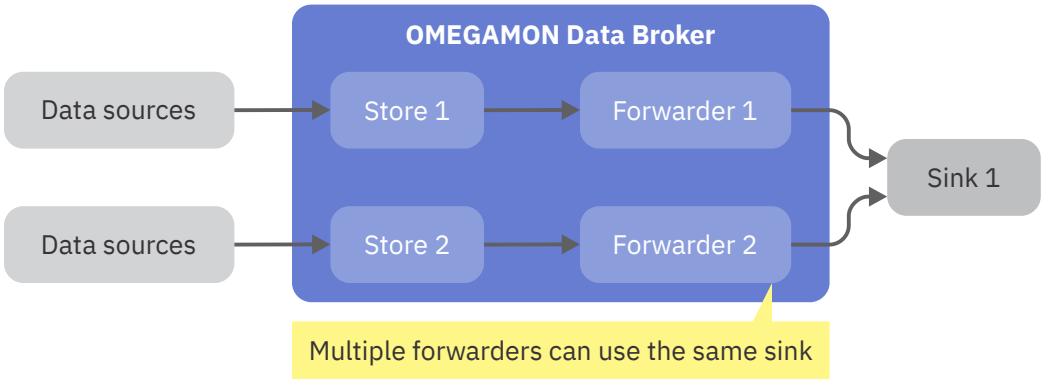


Figure 21. OMEGAMON Data Broker advanced configuration: two stores, two forwarders, one sink

Different data sources can write to different stores. You can choose to forward data from each store to separate instances of OMEGAMON Data Connect or to the same instance of OMEGAMON Data Connect.

If you want to analyze data from multiple sources in the same destination analytics platform, then you might choose to forward data from multiple stores to the same instance of OMEGAMON Data Connect, which then makes all the data available to that analytics platform.

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Stores
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON
KAY.CIDB.STORE.STORE2.NAME=SOURCEB
```

```
* Sink
KAY.CIDB.SINK.SINK1.HOST=analytics1.example.com
KAY.CIDB.SINK.SINK1.PORT=15351

* Forwarders
KAY.CIDB.FWD=ON

KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1

KAY.CIDB.FWD.FWD2.SOURCE_STORE=SOURCEB
KAY.CIDB.FWD.FWD2.SINK=SINK1
```

The following example shows a similar configuration that uses TLS for the connection to OMEGAMON Data Connect. This example demonstrates the usefulness of reusable sink parameters for forwarders that use the same sink: you only have to specify the sink SSL parameters once.

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Stores
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON
KAY.CIDB.STORE.STORE2.NAME=SOURCEB

* Sink
KAY.CIDB.SINK.SINK1.HOST=analytics1.example.com
KAY.CIDB.SINK.SINK1.PORT=15351
KAY.CIDB.SINK.SINK1.FWD1.SECURITY=TLSv1.2
KAY.CIDB.SINK.SINK1.FIPS=ON
KAY.CIDB.SINK.SINK1.KEYRING=KAYSIS01/KAYring

* Forwarders
KAY.CIDB.FWD=ON

KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1

KAY.CIDB.FWD.FWD2.SOURCE_STORE=SOURCEB
KAY.CIDB.FWD.FWD2.SINK=SINK1
```

Example: Forwarding data from each store to a separate instance of OMEGAMON Data Connect

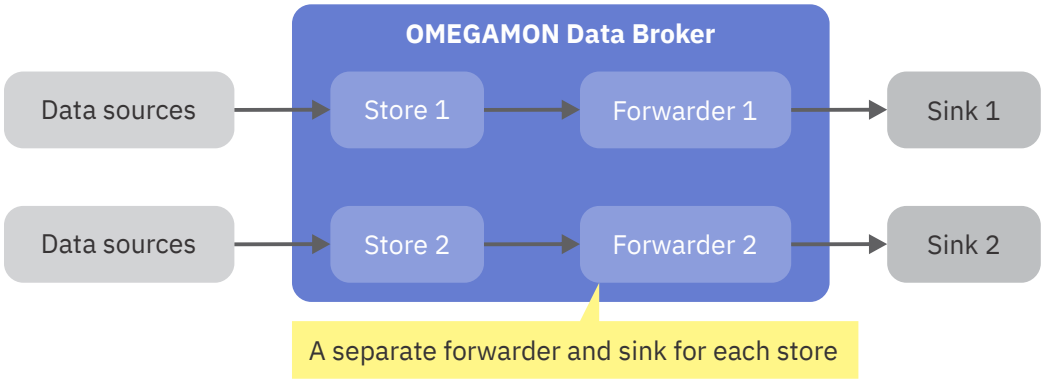


Figure 22. OMEGAMON Data Broker advanced configuration: two stores, two forwarders, two sinks

Different data sources can write to different stores. You can choose to forward data from each store to separate instances of OMEGAMON Data Connect or to the same instance of OMEGAMON Data Connect.

If you want to analyze data from different sources in different destination analytics platforms, then you might choose to forward data from each store to a separate instance of OMEGAMON Data Connect, which then makes the data available to the appropriate analytics platform.

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Stores
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON
KAY.CIDB.STORE.STORE2.NAME=SOURCEB

* Sinks
KAY.CIDB.SINK.SINK1.HOST=analytics1.example.com
KAY.CIDB.SINK.SINK1.PORT=15351

KAY.CIDB.SINK.SINK2.HOST=analytics2.example.com
KAY.CIDB.SINK.SINK2.PORT=15351

* Forwarders
KAY.CIDB.FWD=ON

KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1

KAY.CIDB.FWD.FWD2.SOURCE_STORE=SOURCEB
KAY.CIDB.FWD.FWD2.SINK=SINK2
```

Example: Completing or overriding reusable sink parameters with forwarder-scope sink parameters

If you use a common subset of parameters for different sinks, you can define the common subset as reusable sink parameters, and then use forwarder-scope sink parameters to complete or override those reusable parameters.

For example, suppose you use the same port number and SSL parameters to forward data to OMEGAMON Data Connect on different hosts. You can define reusable sink parameters that include the port number and SSL parameters, but exclude the hostname, and then use forwarder-scope sink parameters to complete the sink parameters by supplying a hostname:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Sink (incomplete: no HOST)
KAY.CIDB.SINK.SINK1.PORT=15351
KAY.CIDB.SINK.SINK1.FWD1.SECURITY=TLSv1.2
KAY.CIDB.SINK.SINK1.FIPS=ON
KAY.CIDB.SINK.SINK1.KEYRING=KAYSIS01/KAYring

* Forwarders
KAY.CIDB.FWD=ON

KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1
KAY.CIDB.FWD.FWD1.SINK_HOST=analytics1.example.com

KAY.CIDB.FWD.FWD2.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD2.SINK=SINK1
KAY.CIDB.FWD.FWD2.SINK_HOST=analytics2.example.com
```

You can also use forwarder-scope sink parameters to override individual reusable sink parameters. In the following example, forwarder 1 uses the hostname specified by the reusable sink parameters, while forwarder 2 overrides that hostname:

```
ZWES.PLUGIN.KAY.ZISDYNAMIC=KAYSISDL
ZWES.PLUGIN.KAY.CIDB=KAYB0001

* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Sink
KAY.CIDB.SINK.SINK1.HOST=analytics1.example.com
KAY.CIDB.SINK.SINK1.PORT=15351
KAY.CIDB.SINK.SINK1.FWD1.SECURITY=TLSv1.2
KAY.CIDB.SINK.SINK1.FIPS=ON
KAY.CIDB.SINK.SINK1.KEYRING=KAYSIS01/KAYring

* Forwarders
KAY.CIDB.FWD=ON

* Uses the hostname specified by the reusable sink parameteters
KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1

* Overrides the hostname specified by the reusable sink parameters
KAY.CIDB.FWD.FWD2.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD2.SINK=SINK1
KAY.CIDB.FWD.FWD2.SINK_HOST=analytics2.example.com
```

Bad example: Multiple forwarders using the same store and the same sink

Do not use this configuration pattern. The sink receives duplicate data.

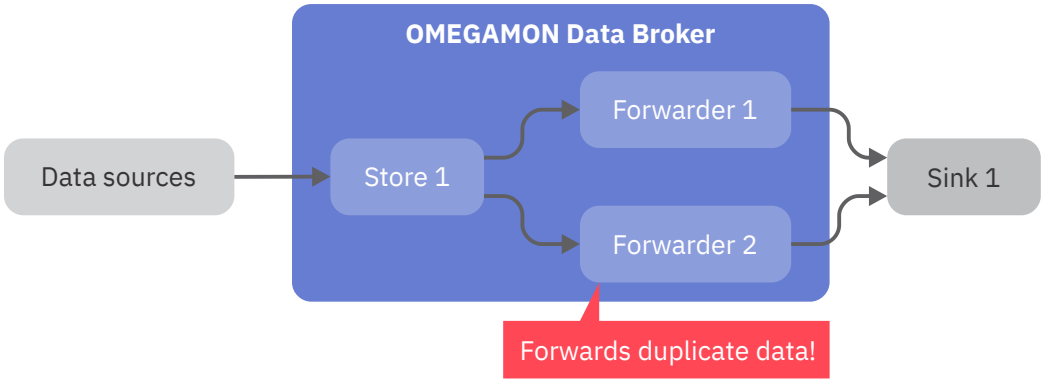


Figure 23. OMEGAMON Data Broker advanced configuration: two forwarders, same store, same sink: do not do this!

Be careful that you do not inadvertently define this bad configuration pattern:

```
* Store
KAY.CIDB.STORE.STORE1.NAME=OMEGAMON

* Sink
KAY.CIDB.SINK.SINK1.HOST=analytics1.example.com
KAY.CIDB.SINK.SINK1.PORT=15351

* Forwarders with same store and sink: do not do this!
KAY.CIDB.FWD=ON

KAY.CIDB.FWD.FWD1.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD1.SINK=SINK1
```

```
KAY.CIDB.FWD.FWD2.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.FWD2.SINK=SINK1
```

OMEGAMON Data Connect configuration parameters

OMEGAMON Data Connect configuration parameters identify inputs, such as the TCP port on which to listen for data from OMEGAMON Data Broker, and outputs, such as destination analytics platforms. You can filter which data to output.

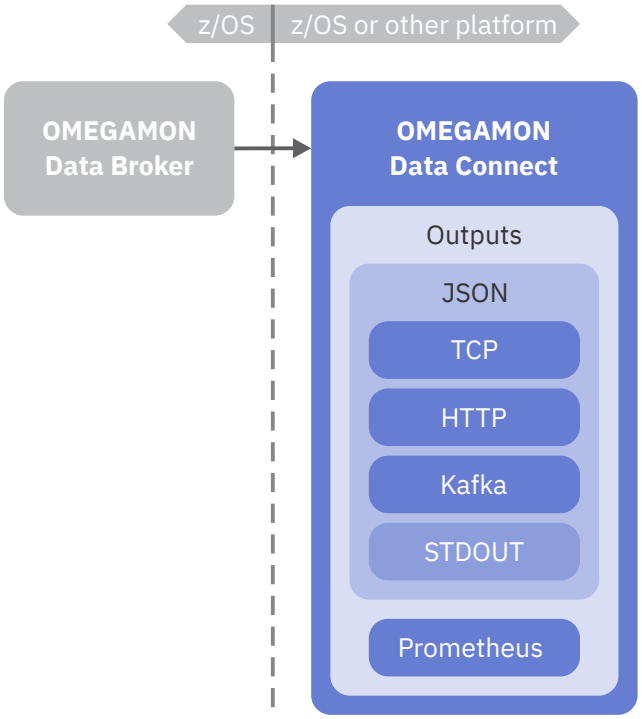


Figure 24. OMEGAMON Data Connect configuration points: input from OMEGAMON Data Broker and various outputs

Format

The OMEGAMON Data Connect configuration file, `connect.yaml`, is a YAML document. OMEGAMON Data Connect configuration parameters and their values conform to YAML syntax.

Here is the high-level structure of the document. Lower-level structures are indicated by placeholder labels inside angle brackets (< >):

```
connect:
  <Connect-specific parameters>

# Common Spring Boot application properties

server: # Optional
  <Spring Boot server properties>

logging: # Optional
  <Spring Boot logging properties>
```

Tip: To check your configuration parameters, use the OMEGAMON Data Provider configuration validator.

`connect` is the parent key for parameters that are specific to OMEGAMON Data Connect.

OMEGAMON Data Connect is a Java application developed using the Spring Boot framework. In addition to Connect-specific parameters, the OMEGAMON Data Connect configuration file can also

specify common Spring Boot properties. The following keys set common Spring Boot properties that are particularly relevant to OMEGAMON Data Connect:

`server`
The parent key for Spring Boot server properties.

`logging`
The parent key for Spring Boot logging properties.

For more details on common Spring Boot properties, see the Spring Boot documentation on common application properties.

Terminology: parameter versus property

In the context of OMEGAMON Data Connect, the terms *parameter* and *property* are interchangeable synonyms. In general, OMEGAMON Data Provider uses the term *configuration parameter* or just *parameter* across all components, whereas Spring Boot uses the term *property*. For consistency with Spring Boot documentation, for artifacts that are already defined by Spring Boot, OMEGAMON Data Provider documentation uses the term *property*.

Character encoding

The configuration file must be encoded in UTF-8.

If the file is not valid UTF-8, then OMEGAMON Data Connect reports the error `java.nio.charset.MalformedInputException` and stops.

Location

OMEGAMON Data Connect configuration parameters are stored in a `config/connect.yaml` file in an OMEGAMON Data Connect user directory.

A sample configuration file is supplied in `config/connect.yaml` in the OMEGAMON Data Connect installation directory..

Dot notation for YAML parameters

Some references to YAML configuration parameters use dot notation as a concise method for indicating the parameter hierarchy. Dot notation is not for direct use in the YAML document.

For example, `connect.output.prometheus.mappings` represents the following YAML hierarchy:

```
connect:
  output:
    prometheus:
      mappings:
```

Example: Output to JSON Lines over TCP without SSL/TLS

This example configures OMEGAMON Data Connect with the following behavior:

- Receive input from OMEGAMON Data Broker over TCP on port 15351 of the local z/OS host.
- Send output in JSON Lines format over TCP to a remote host named `elastic.example.com` on which Logstash has been configured to listen on port 5046.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: localhost
      port: 15351
  output:
```

```
    tcp:
      enabled: true
      sinks:
        logstash:
          hostname: elastic.example.com
          port: 5046
```

For more examples, including secure (SSL/TLS) configuration examples, see the topics on each input and output method.

Related tasks

Configuring OMEGAMON Data Connect
OMEGAMON Data Connect is a Java application that can run on or off z/OS.

Reloading OMEGAMON Data Connect configuration
After updating the contents of an OMEGAMON Data Connect user directory, such as the `connect.yaml` configuration file or mapping extension JAR files, you need to apply those changes to the instances of OMEGAMON Data Connect that refer to the user directory. To apply the changes, you need to restart (stop and then start) the affected instances of OMEGAMON Data Connect.

Related reference

OMEGAMON Data Connect user directory
An OMEGAMON Data Connect user directory contains files that configure OMEGAMON Data Connect for your site.

Connect-specific parameters

The **connect** key sets parameters that are specific to OMEGAMON Data Connect.

```
connect:
  input:
    tcp: # Required
      <TCP input parameters>

  output: # At least one output is required
    tcp:
      <TCP output parameters>

    http:
      <HTTP output parameters>

    kafka:
      <Kafka output parameters>

    prometheus:
      <Prometheus output parameters>

    stdout:
      <STDOUT output parameters>

  filter: # Optional
    <Global-level filter for JSON outputs>

  event-publisher: # Optional
    <Event publisher parameters>

  logging: # Optional
    <Connect-specific logging parameters>
```

input
OMEGAMON Data Connect supports a single input: data from OMEGAMON Data Broker over TCP.

output
A single instance of OMEGAMON Data Connect can send data to all of these outputs:

tcp
JSON Lines over TCP. You can specify multiple destinations ("sinks") for TCP output.

http
JSON in HTTP POST requests. You can specify multiple destinations ("endpoints") for HTTP output.

kafka
JSON published to Apache Kafka. You can publish either to a single topic, or to a separate topic for each table (for example, each OMEGAMON attribute group).

prometheus
Prometheus metrics HTTP endpoint hosted by OMEGAMON Data Connect.

stdout
JSON Lines written to the `stdout` file.

filter
Filters which tables (such as OMEGAMON attribute groups) and which fields (such as OMEGAMON attributes) from those tables to send to the JSON-format outputs: `tcp`, `http`, `kafka`, and `stdout`.

event-publisher
Controls aspects of internal OMEGAMON Data Connect processing.

logging
Controls logging behavior specific to OMEGAMON Data Connect, such as flood control.

TCP input parameters

OMEGAMON Data Connect TCP input parameters specify how OMEGAMON Data Connect listens for data over a TCP network from OMEGAMON Data Broker.

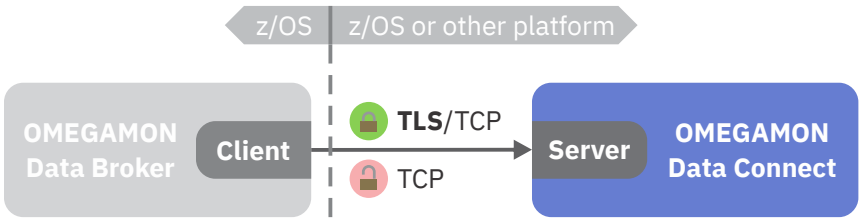


Figure 25. OMEGAMON Data Connect configuration: TCP input

In the context of OMEGAMON Data Connect receiving data from OMEGAMON Data Broker, OMEGAMON Data Connect is the *server* and OMEGAMON Data Broker is the *client*.

```
connect:
  input:
    tcp:
      enabled: <true|false> # Default at this level: false
      hostname: <string>
      port: <number>
      ssl: # Optional
        <SSL parameters>
```

enabled
Whether this function is enabled. Allowed values: `true`, `false`. This key is optional. Default: `false`.

To enable this function, you must specify `enabled: true`.

Specifying `enabled: false` has the same effect as commenting-out the parent key of this `enabled` key and all descendants of that parent key.

hostname
Hostname or IP address on which the OMEGAMON Data Connect host listens for data from OMEGAMON Data Broker.

Some typical values:

0.0.0.0
All IPv4 addresses on the local machine.

localhost or 127.0.0.1
Loopback address.

This hostname or IP address must correspond to the hostname or IP address to which OMEGAMON Data Broker sends data, specified by the OMEGAMON Data Broker parameter SINK_HOST.

If you run OMEGAMON Data Connect and OMEGAMON Data Broker on the same z/OS instance (LPAR), then you can specify `localhost` for both this hostname and the OMEGAMON Data Broker sink hostname.

port
Port on which to listen for data from OMEGAMON Data Broker.

This value must match the OMEGAMON Data Broker parameter SINK_PORT.

SSL parameters

`connect.input.tcp.ssl`:

```
enabled: <true|false>
ciphers: <ciphers_list>
client-auth: <need|none|want>
enabled-protocols: <protocols_list>
protocol: <protocol>
key-alias: <string>
key-password: <string>
key-store: <string>
key-store-password: <string>
key-store-type: <JKS|PKCS12|JCERACFKS>
trust-store: <string>
trust-store-password: <string>
trust-store-type: <JKS|PKCS12|JCERACFKS>
```

enabled
Whether to enable SSL/TLS:

true
Enable SSL/TLS.

false
Disable SSL/TLS.

This key is optional. Default: `true`.

Use `enabled: false` as a convenient single-line method for disabling SSL/TLS, as an alternative to using YAML comment syntax to comment-out all of the SSL parameters.

ciphers
A list of candidate ciphers for the connection, in one of the following formats:

- OpenSSL cipher list
- A comma-separated list of ciphers using the standard OpenSSL cipher names or the standard JSSE cipher names

This key is optional. Example, in OpenSSL cipher list format:

`HIGH:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!kRSA`

client-auth
Client authentication. Whether to request a client certificate from the client, and then whether to allow the connection based on the client response.

need
Request a client certificate. Allow the connection only if the client responds with a valid certificate.

none
Do not request a client certificate. Allow the connect without client authentication.

want
Request a client certificate. If the client responds with a certificate, allow the connection only if the certificate is valid. If the client does not respond with a certificate, allow the connection.

enabled-protocols
List of protocols to enable.

This key is optional. Example:

```
TLSv1.3,TLSv1.2
```

protocol
Protocol to use.

If this protocol is not supported by both ends of the connection, then the connection can fall back (downgrade) to one of the other enabled protocols.

This key is optional. Default in Java 17: `TLSv1.3`.

key-alias
Alias of the server private key and associated server certificate in the keystore. On z/OS, the alias is also known as the certificate *label*.

This key is optional. Default: the default certificate in the keystore.

key-password
Password required to access the server private key in the keystore.

This key is optional. Default: the value of `key-store-password`.

key-store-password
Password to access the keystore.

If the keystore type is `JCERACFKS`, then specify the fixed value:

```
password
```

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

key-store
Location of the keystore that contains the server certificate.

The location format depends on the keystore type:

JKS
Keystore file path. Example:

```
/u/my/security/certs/certs.jks
```

PKCS12
Keystore file path. Example:

```
/u/my/security/certs/certs.p12
```

JCERACFKS
Only valid if OMEGAMON Data Connect runs on z/OS.

RACF key ring, in the following format:

```
safkeyring://<owner_user_id>/<key_ring_name>
```

Note: In this specific context, follow `safkeyring:` with two (2) consecutive slashes.

where *<owner_user_id>* is the RACF user ID that owns the key ring and *<key_ring_name>* is the RACF key ring name. Example:

```
safkeyring://STCOMDP/OMDPring
```

key-store-type
Keystore type. Supported types depend on the security providers in the JRE. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore (*key ring*). Only available if OMEGAMON Data Connect is running on z/OS and the IBMZSecurity provider is available in the JRE.

trust-store
Location of the truststore that contains trusted client certificates. See the list of example locations for `key-store`.

A truststore is required only for client authentication; that is, when the value of `client-auth` is `need` or `want`.

trust-store-password
Password to access the truststore.

If the truststore type is JCERACFKS, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

trust-store-type
Truststore type. See the list of example types for `key-store-type`.

Example: Connection to OMEGAMON Data Broker running on the same z/OS LPAR as OMEGAMON Data Connect, with no TLS

This example listens for data from OMEGAMON Data Broker on the localhost loopback address. In this case, OMEGAMON Data Connect and OMEGAMON Data Broker must be running on the same z/OS instance (LPAR). The data is not exposed on a network; there is no TLS.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: localhost
      port: 15379

  output:
    # One or more outputs...
```

Example: Secure connection over TLS using the same RACF key ring as both keystore and truststore

In this example:

- OMEGAMON Data Connect is running on z/OS, so it can use the JCERACFKS keystore and truststore type, and refer to RACF key rings. Note the fixed value `password` for the keystore and truststore passwords.

- OMEGAMON Data Connect requires client authentication: OMEGAMON Data Broker must provide a valid certificate.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379
      ssl:
        enabled-protocols: TLSv1.2
        protocol: TLS
        client-auth: need
        # Certificates of trusted clients (instances of OMEGAMON Data Broker)
        trust-store: safkeyring://STCOMDP/OMDPring
        trust-store-type: JCERACFKS
        trust-store-password: password
        # Server certificate
        key-store: safkeyring://STCOMDP/OMDPring
        key-store-type: JCERACFKS
        key-store-password: password
        key-alias: OMDPcert

  output:
    # One or more outputs...
```

Example: Secure connection over TLS using PKCS12 keystore and JKS truststore

In this example:

- OMEGAMON Data Connect might be running on or off z/OS.
- OMEGAMON Data Connect requires client authentication: OMEGAMON Data Broker must provide a valid certificate.
- OMEGAMON Data Connect uses the default certificate in the keystore.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379
      ssl:
        enabled-protocols: TLSv1.2
        protocol: TLS
        client-auth: need
        # Trusted client certificates
        trust-store: /u/my/security/certs/omdp-broker.jks
        trust-store-type: JKS
        trust-store-password: Pa$$w0rdTS
        # Server certificate
        key-store: /u/my/security/certs/omdp-connect.p12
        key-store-type: PKCS12
        key-store-password: Pa$$w0rdKS

  output:
    # One or more outputs...
```

Related reference

OMEGAMON Data Broker configuration parameters

OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

TCP output parameters

OMEGAMON Data Connect TCP output parameters specify one or more destinations ("sinks") for sending data in JSON Lines format over a TCP network.

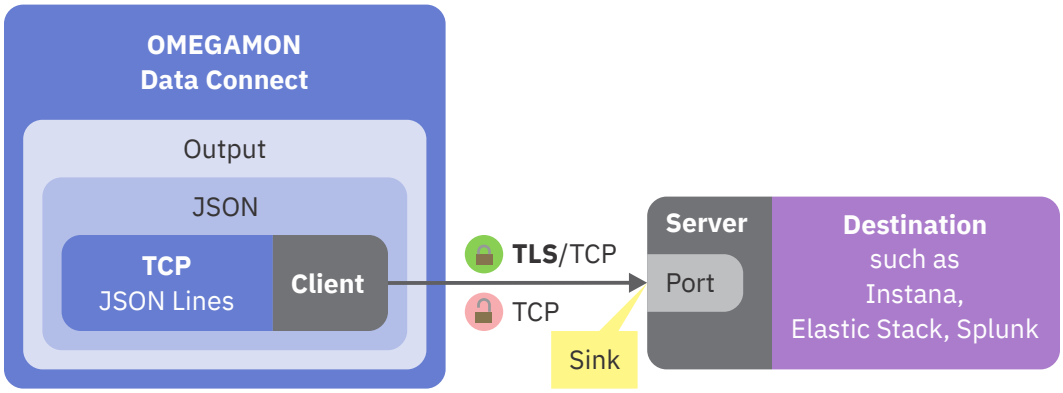


Figure 26. OMEGAMON Data Connect configuration: TCP output

In the context of OMEGAMON Data Connect sending data over TCP, OMEGAMON Data Connect is the *client* and the destination is the *server*.

```
connect:
  output:
    tcp:
      enabled: <true|false> # Default at this level: false
      sinks: # One or more sinks (destinations)
        <sink_name_1>: # Each sink has a unique name of your choice
          enabled: <true|false> # Default at this level: true
          hostname: <string>
          port: <number>
          # All of the following parameters are optional
          max-connection-attempts: <number>
          retry-interval: <seconds>
          ssl:
            <SSL parameters>
          filter: # Output-level filter
            <Filter parameters>
        <sink_name_2>: # Additional sink
          ...
```

enabled

Whether this function is enabled. Allowed values: `true`, `false`. This key is optional.

You can specify the `enabled` key as a child of the `tcp` key and as a child of each *<sink_name>*.

Defaults:

```
connect.output.tcp.enabled: false
connect.output.tcp.sinks.<sink_name>.enabled: true
```

Specifying `enabled: false` has the same effect as commenting-out the parent key of the `enabled` key and all descendants of that parent key.

To enable *any* sinks, you must specify `connect.output.tcp.enabled: true`.

To disable a sink, specify `connect.output.tcp.sinks.<sink_name>.enabled: false`.

To disable all sinks, either omit `connect.output.tcp.enabled` or specify `connect.output.tcp.enabled: false`.

<sink_name_1>, <sink_name_2>, ...
OMEGAMON Data Connect can send to multiple sinks.

Sink names are your choice. You might choose descriptive names, such as `logstash` and `splunk`. See the examples at the end of this topic.

hostname
Destination hostname or IP address on which software is listening for JSON Lines over TCP.

port
Destination port.

max-connection-attempts
Optional. Maximum number of attempts to connect to the sink. Default: no value; unlimited.

OMEGAMON Data Connect attempts to connect to the sink in two situations:

- When OMEGAMON Data Connect starts.
- When the connection is lost.

To avoid unlimited connection attempts, set a `max-connection-attempts` value.

retry-interval
Optional. Number of seconds to wait before retrying connection to the sink, either when attempting initial connection at startup or when the connection is lost. Default: 20.

filter
Optional filter to restrict what data to send.

This output-level filter applies only to this sink, replacing any global-level filter (`connect.filter`).

Tip: You can specify an output-level filter for each sink (`connect.output.tcp.sinks.<sink_name>.filter`) and a global-level filter that applies to all JSON-format outputs (`connect.filter`). However, you cannot specify a filter that applies *only to all TCP outputs*; there is no `connect.output.tcp.filter`.

SSL parameters

`connect.output.tcp.sinks.<sink_name>.ssl`:

```
enabled: <true|false>
ciphers: <ciphers_list>
enabled-protocols: <protocols_list>
protocol: <protocol>
key-alias: <string>
key-password: <string>
key-store: <string>
key-store-password: <string>
key-store-type: <JKS|PKCS12|JCERACFKS>
trust-store: <string>
trust-store-password: <string>
trust-store-type: <JKS|PKCS12|JCERACFKS>
```

enabled
Whether to enable SSL/TLS:

true
Enable SSL/TLS.

false
Disable SSL/TLS.

This key is optional. Default: `true`.

Use `enabled: false` as a convenient single-line method for disabling SSL/TLS, as an alternative to using YAML comment syntax to comment-out all of the SSL parameters.

ciphers
A list of candidate ciphers for the connection, in one of the following formats:

- OpenSSL cipher list
- A comma-separated list of ciphers using the standard OpenSSL cipher names or the standard JSSE cipher names

This key is optional. Example, in OpenSSL cipher list format:

```
HIGH:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!kRSA
```

enabled-protocols
List of protocols to enable.

This key is optional. Example:

```
TLSv1.3,TLSv1.2
```

protocol
Protocol to use.

If this protocol is not supported by both ends of the connection, then the connection can fall back (downgrade) to one of the other enabled protocols.

This key is optional. Default in Java 17: `TLSv1.3`.

key-alias
Alias of the client private key and associated client certificate in the keystore. On z/OS, also known as the certificate *label*.

This key is optional. Default: the default certificate in the keystore.

key-password
Password required to access the client private key in the keystore.

This key is optional. Default: the value of `key-store-password`.

key-store-password
Password to access the keystore.

If the keystore type is JCERACFKS, then specify the fixed value:

```
password
```

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

key-store
Location of the keystore that contains the client certificate.

A keystore is required only if the server requires client authentication.

The location format depends on the keystore type:

JKS
Keystore file path. Example:

```
/u/my/security/certs/keystore.jks
```

PKCS12
Keystore file path. Example:

```
/u/my/security/certs/keystore.p12
```

JCERACFKS
Only valid if OMEGAMON Data Connect runs on z/OS.

RACF key ring, in the following format:

safkeyring://*<owner_user_id>*/*<key_ring_name>*

Note: In this specific context, follow `safkeyring:` with two (2) consecutive slashes.

where *<owner_user_id>* is the RACF user ID that owns the key ring and *<key_ring_name>* is the RACF key ring name.

key-store-type
Keystore type. Supported types depend on the security providers in the JRE. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore (*key ring*). Only available if OMEGAMON Data Connect is running on z/OS and the IBMZSecurity provider is available in the JRE.

trust-store
Location of the truststore that contains trusted server certificates. See the list of example locations for `key-store`.

trust-store-password
Password to access the truststore.

If the truststore type is `JCERACFKS`, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

trust-store-type
Truststore type. See the list of example types for `key-store-type`.

Example: Connection without TLS

```
connect:
  input: # From OMEGAMON Data Broker...
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    tcp:
      enabled: true # Required to enable any sinks: default is false
      sinks:
        splunk:
          enabled: true # Optional: default is true
          hostname: splunk.example.com
          port: 5046
```

Example: Multiple destinations

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    tcp:
      enabled: true
      sinks:
```

```
        logstash1: # Descriptive sink name
          hostname: elastic1.example.com
          port: 5046
        logstash2:
          hostname: elastic2.example.com
          port: 5046
        splunk:
          hostname: splunk.example.com
          port: 5047
```

Example: Secure connection over TLS with client authentication, using the same RACF key ring as both keystore and truststore

In this example:

- OMEGAMON Data Connect is running on z/OS, so it can use the JCERACFKS keystore and truststore type, and refer to RACF key rings.
- The destination server, Logstash, requires client authentication, so the SSL parameters here include client certificate details: the keystore and key alias (in RACF terms, the certificate *label*).

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    tcp:
      enabled: true
      sinks:
        logstash:
          hostname: elastic.example.com
          port: 5046
          ssl:
            enabled: true
            enabled-protocols: TLSv1.2
            protocol: TLS
            trust-store: safkeyring://STCOMDP/OMDPring
            trust-store-type: JCERACFKS
            trust-store-password: password

            # If Logstash requires client authentication
            key-store: safkeyring://STCOMDP/OMDPring
            key-store-type: JCERACFKS
            key-store-password: password
            key-alias: Cert.OMDP
```

Example: Secure connection over TLS with client authentication, using PKCS12 keystore and JKS truststore

In this example, OMEGAMON Data Connect might be running on or off z/OS: the SSL parameter values are not z/OS-specific.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
```

```
    tcp:
      enabled: true
      sinks:
        logstash:
          hostname: elastic.example.com
          port: 5046
          ssl:
            enabled-protocols: TLSv1.2
            protocol: TLS
            # Server certificates
            trust-store: /u/my/security/certs/omdp-connect-sinks.jks
            trust-store-type: JKS
            trust-store-password: Pa$$w0rdTS
            # Client certificate
            key-store: /u/my/security/certs/omdp-connect.p12
            key-store-type: PKCS12
            key-store-password: Pa$$w0rdKS
```

Related reference

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Characteristics of JSON output from OMEGAMON Data Connect
If you need to work directly with JSON output from OMEGAMON Data Connect, then it's useful to understand the characteristics of this data, such as its structure, property names, and property values.

HTTP output parameters

OMEGAMON Data Connect HTTP output parameters specify one or more destinations ("endpoints") for sending data in JSON format in an HTTP/1.1 POST request.

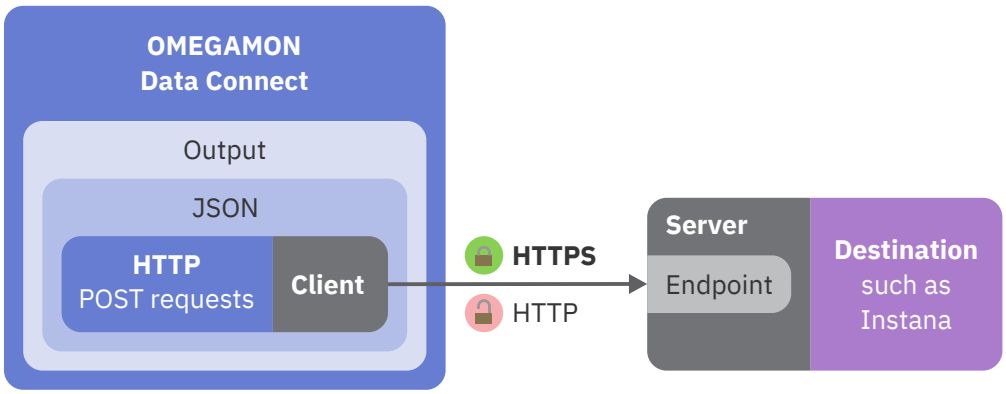


Figure 27. OMEGAMON Data Connect configuration: HTTP output

In the context of OMEGAMON Data Connect sending data over HTTP, OMEGAMON Data Connect is the *client* and the destination is the *server*.

```
connect:
  output:
    http:
      enabled: <true|false> # Default at this level: false
      endpoints: # One or more endpoints (destinations)
        <endpoint_name_1>: # Each endpoint has a unique name of your choice
          enabled: <true|false> # Default at this level: true
          url: <string>
          # All of the following parameters are optional
          call-timeout: <seconds>
          connect-timeout: <seconds>
          write-timeout: <seconds>
          read-timeout: <seconds>
```

```
          max-failures: <number>
          compression: <true|false> # Default: false
          headers: # One or more custom headers to add to each request
            - key: <header_key_1>
              value: <header_value_1>
            - key: <header_key_2>
              value: <header_value_2>
          batching: # Send multiple records per request
            enabled: <true|false> # Default: false (single record per
request)
            # batch-size and linger apply only when batching is enabled
            batch-size: <number_of_records> # Default: 1000
            linger: <duration> # Default: 250ms
          ssl:
            <SSL parameters>
          filter: # Output-level filter
            <Filter parameters>
        <endpoint_name_2>: # Additional endpoint
          ...
```

enabled
Whether this function is enabled. Allowed values: `true`, `false`. This key is optional.

You can specify an `enabled` key in various contexts. The default value depends on the context. Example defaults:

```
connect.output.http.enabled: false
connect.output.http.endpoints.<endpoint_name>.enabled: true
connect.output.http.endpoints.<endpoint_name>.batching.enabled: false
```

Specifying `enabled: false` has the same effect as commenting-out the parent key of the `enabled` key and all descendants of that parent key.

To enable sending to *any* endpoints, you must specify `connect.output.http.enabled: true`.

To disable sending to an endpoint, specify `connect.output.http.endpoints.<endpoint_name>.enabled: false`.

To disable sending to all endpoints, either omit `connect.output.http.enabled` or specify `connect.output.http.enabled: false`.

<endpoint_name_1>, <endpoint_name_2>, ...
OMEGAMON Data Connect can send to multiple endpoints.

Endpoint names are your choice. You might choose descriptive names, such as `instana`. See the examples at the end of this topic.

url
HTTP or HTTPS URL of the destination endpoint. If the URL does not specify a port, then the default port is 80 for HTTP or 443 for HTTPS.

If you specify an HTTPS URL, then you must also specify SSL parameters.

call-timeout
Optional. Either:

- Integer number of seconds to allow for a request to complete before considering the request a failure
- 0 (zero), meaning unlimited; no timeout

Default: 0 (no timeout).

The call timeout covers the entire duration of an HTTP request, ending when OMEGAMON Data Connect receives the complete response from the server. Other timeout parameters cover individual stages of a request.

A call timeout contributes to the failures counted by `max-failures`.

connect-timeout
Optional. Either:

- Integer number of seconds to allow for establishing a connection to the server before considering the request a failure
- 0 (zero), meaning unlimited; no timeout

Default: 10.

OMEGAMON Data Connect reuses connections. The connect timeout applies only to an HTTP request that requires a new connection.

write-timeout
Optional. Either:

- Integer number of seconds to allow for writing the request to the server before considering the request a failure
- 0 (zero), meaning unlimited; no timeout

Default: 10.

One possible cause of write timeout is an unusually large request body.

read-timeout
Optional. Either:

- Integer number of seconds to allow for reading a response from the server before considering the request a failure
- 0 (zero), meaning unlimited; no timeout

Default: 10.

The elapsed time to read a complete response can exceed the read timeout. A response can arrive in a single piece or in multiple pieces spread over time. The read timeout applies to each attempt to read a piece of the response.

Tip: There is no separate timeout parameter for the complete response. To indirectly set a timeout for the complete response, set `call-timeout`, which applies to the entire HTTP request, including the time to read the complete response.

max-failures
Optional. Maximum number of failures to allow before stopping sending requests to the endpoint. Default: no value; unlimited.

Possible failures include timeouts, connection failures, and unsuccessful responses.

To avoid unlimited failures, set a `max-failures` value. For example:

- To not allow any failures, and stop sending requests to the endpoint immediately after the first failure, set `max-failures` to 0 (zero)
- To allow 5 failures, and then stop sending requests to the endpoint if a 6th failure occurs, set `max-failures` to 5

compression
Optional. Whether to gzip-compress the request body. Allowed values: `true`, `false`. Default: `false`; the request body is not compressed.

headers
Optional. Sequence of custom header key-value pairs to add to each request.

For example, the following `headers` parameter:

```
headers:
  - key: odp_header_key_1
    value: header_value_1
```

```
  - key: odp_header_key_2
    value: header_value_2
```

adds the following headers to each request:

```
odp_header_key_1: header_value_1
odp_header_key_2: header_value_2
```

Some analytics platforms use custom headers to determine how to process a request. For details, see the documentation for your analytics platform.

Don't set headers that OMEGAMON Data Connect already sets:

`Accept-Encoding`
`Connection`
`Content-Encoding`
`Content-Length`
`Content-Type`
`Host`
`Transfer-Encoding`
`odp-proto`
`User-Agent`

batching

Optional. When batching is enabled, each request can contain multiple records. Batching offers higher throughput than the default single-record-per-request behavior.

enabled

Whether to enable batching:

true

Enable batching.

Each request contains up to `batch-size` records, according to the following rules:

- If the number of records in the queue is at least `batch-size`, then the output sends a request containing `batch-size` records.
- If there are fewer than `batch-size` records in the queue, then the output waits for up to the duration specified by `linger`. If the number of records in the queue reaches `batch-size` within that duration, then the output sends a request containing `batch-size` records. Otherwise, if there are still fewer than `batch-size` records in the queue at the end of that duration, then the output sends a request containing however many records are in the queue.

Each record in the request body is represented by a length-prefixed JSON object. Each JSON object is prefixed by a 4-byte binary-encoded integer, in network byte order, that states the length of the JSON object.

Tip: Before enabling batching, check whether the destination analytics platform supports HTTP POST request bodies that contain a stream of length-prefixed JSON objects, with the length stated in the specific format described here.

Requests contain the header `odp-proto: batch`.

false

Disable batching.

Each request contains only a single record.

The request body consists of a single JSON object representing a single record. By contrast with the request body format when batching is enabled, this single JSON object has no length prefix.

Requests contain the header `odp-proto: single`.

This key is optional. Default: `false` (single record per request).

`batch-size`
Optional; applies only if batching is enabled. Maximum number of records to send in a request. Default: `1000`.

Tip: Before setting a large `batch-size`, check whether the destination HTTP server will accept the corresponding large requests. Request size is determined by the combination of the number of records in the request and the record lengths. Record lengths can vary depending on factors such as the record types that you have selected to send: different record types involve different numbers of fields and different field lengths. HTTP servers typically have a configuration parameter that sets a maximum request size limit, and reject requests that exceed that limit. For details, see the HTTP server configuration parameter documentation for your destination analytics platform.

`linger`
Optional; applies only if batching is enabled. Duration to wait if the number of records in the queue is less than `batch-size`.

Allowed values: a positive integer optionally followed by any of these units:

`ns`
Nanoseconds

`us`
Microseconds

`ms`
Milliseconds

`s`
Seconds

`m`
Minutes

`h`
Hours

`d`
Days

For example, the value `5m` means a duration of 5 minutes.

If the unit is omitted, seconds are used. For example, `2` and `2s` are equivalent.

Default: `250ms`.

`filter`
Optional filter to restrict what data to send.

This output-level filter applies only to this endpoint, replacing any global-level filter (`connect.filter`).

Tip: You can specify an output-level filter for each endpoint (`connect.output.http.endpoints.<endpoint_name>.filter`) and a global-level filter that applies to all JSON-format outputs (`connect.filter`). However, you cannot specify a filter that applies *only to all HTTP outputs*; there is no `connect.output.http.filter`.

SSL parameters

`connect.output.http.endpoints.<endpoint_name>.ssl`:

```
enabled: <true|false>
ciphers: <ciphers_list>
enabled-protocols: <protocols_list>
protocol: <protocol>
key-alias: <string>
```

```
key-password: <string>
key-store: <string>
key-store-password: <string>
key-store-type: <JKS|PKCS12|JCERACFKS>
trust-store: <string>
trust-store-password: <string>
trust-store-type: <JKS|PKCS12|JCERACFKS>
```

enabled
Whether to enable SSL/TLS:

true
Enable SSL/TLS.

false
Disable SSL/TLS.

This key is optional. Default: `true`.

Use `enabled: false` as a convenient single-line method for disabling SSL/TLS, as an alternative to using YAML comment syntax to comment-out all of the SSL parameters.

ciphers
A list of candidate ciphers for the connection, in one of the following formats:

- OpenSSL cipher list
- A comma-separated list of ciphers using the standard OpenSSL cipher names or the standard JSSE cipher names

This key is optional. Example, in OpenSSL cipher list format:

`HIGH:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!kRSA`

enabled-protocols
List of protocols to enable.

This key is optional. Example:

`TLSv1.3,TLSv1.2`

protocol
Protocol to use.

If this protocol is not supported by both ends of the connection, then the connection can fall back (downgrade) to one of the other enabled protocols.

This key is optional. Default in Java 17: `TLSv1.3`.

key-alias
Alias of the client private key and associated client certificate in the keystore. On z/OS, also known as the certificate *label*.

This key is optional. Default: the default certificate in the keystore.

key-password
Password required to access the client private key in the keystore.

This key is optional. Default: the value of `key-store-password`.

key-store-password
Password to access the keystore.

If the keystore type is `JCERACFKS`, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

key-store
Location of the keystore that contains the client certificate.

A keystore is required only if the server requires client authentication.

The location format depends on the keystore type:

JKS
Keystore file path. Example:

`/u/my/security/certs/keystore.jks`

PKCS12
Keystore file path. Example:

`/u/my/security/certs/keystore.p12`

JCERACFKS
Only valid if OMEGAMON Data Connect runs on z/OS.

RACF key ring, in the following format:

`safkeyring://<owner_user_id>/<key_ring_name>`

Note: In this specific context, follow `safkeyring:` with two (2) consecutive slashes.

where *<owner_user_id>* is the RACF user ID that owns the key ring and *<key_ring_name>* is the RACF key ring name.

key-store-type
Keystore type. Supported types depend on the security providers in the JRE. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore (*key ring*). Only available if OMEGAMON Data Connect is running on z/OS and the IBMZSecurity provider is available in the JRE.

trust-store
Location of the truststore that contains trusted server certificates. See the list of example locations for `key-store`.

trust-store-password
Password to access the truststore.

If the truststore type is `JCERACFKS`, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

trust-store-type
Truststore type. See the list of example types for `key-store-type`.

Example: HTTP with call timeout and limited allowed failures

```
connect:
  input: # From OMEGAMON Data Broker...
    tcp:
      enabled: true
      hostname: localhost # on same z/OS instance as OMEGAMON Data Connect
      port: 15379

  output:
```

```
    http:
      enabled: true # Required to enable any endpoints: default is false
      endpoints:
        instana:
          enabled: true # Optional: default is true
          url: http://instana.example.com/endpoint
          call-timeout: 30 # Each request must complete within 30 seconds
          max-failures: 5 # Allow 5 failures before stopping sending to this
endpoint
```

Example: Batched requests with compression

This example sends requests that contain up to 5000 records, gzip-compressed.

If there are fewer than 5000 records in the queue, then the output waits ("lingers") for up to 2 seconds before sending the next request. If the number of records in the queue reaches 5000 within that 2 seconds, then the output sends a request containing 5000 records. Otherwise, if there are still fewer than 5000 records after 2 seconds, then the output sends a request containing however many records are in the queue.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    http:
      enabled: true
      endpoints:
        instana:
          url: http://instana.example.com/endpoint
          compression: true
          batching:
            enabled: true
            batch-size: 5000
            linger: 2s
```

Example: Secure (HTTPS) connection with client authentication, using the same RACF key ring as both keystore and truststore

In this example:

- OMEGAMON Data Connect is running on z/OS, so it can use the JCERACFKS keystore and truststore type, and refer to RACF key rings.
- The destination server requires client authentication, so the SSL parameters here include client certificate details: the keystore and key alias (in RACF terms, the certificate *label*).

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    http:
      enabled: true
      endpoints:
        instana:
          url: https://instana.example.com/endpoint
```

```
          ssl:
            enabled: true
            enabled-protocols: TLSv1.2
            protocol: TLS
            # Server certificates
            trust-store: safkeyring://STCOMDP/OMDPring
            trust-store-type: JCERACFKS
            trust-store-password: password
            # Client certificate
            key-store: safkeyring://STCOMDP/OMDPring
            key-store-type: JCERACFKS
            key-store-password: password
            key-alias: Cert.OMDP
```

Example: Secure (HTTPS) with client authentication, using PKCS12 keystore and JKS truststore

In this example, OMEGAMON Data Connect might be running on or off z/OS.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    http:
      enabled: true
      endpoints:
        instana:
          url: https://instana.example.com/endpoint
          ssl:
            enabled-protocols: TLSv1.2
            protocol: TLS
            trust-store: /u/my/security/certs/omdp-connect-endpoints.jks
            trust-store-type: JKS
            trust-store-password: Pa$$w0rdTS
            key-store: /u/my/security/certs/omdp-connect.p12
            key-store-type: PKCS12
            key-store-password: Pa$$w0rdKS
```

Related reference

Filters for JSON-format outputs

You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Kafka output parameters

OMEGAMON Data Connect Kafka output parameters specify whether to publish data in JSON format to an Apache Kafka topic.

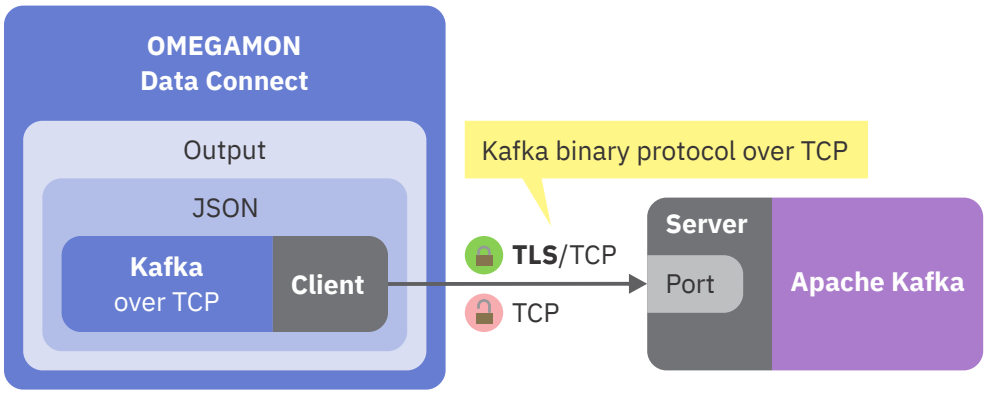


Figure 28. OMEGAMON Data Connect configuration: Kafka output

In this context, OMEGAMON Data Connect is a Kafka *client*. More specifically, OMEGAMON Data Connect is a Kafka *producer*.

```
connect:
  output:
    kafka:
      enabled: <true|false> # Default at this level: false
      servers: <string>
      # All of the following parameters are optional
      topic: <topic_name> # Default: per-table topics
      topic-prefix: <topic_prefix> # Default: odp
      filter: # Output-level filter
        <Filter parameters>
      properties:
        <Kafka producer configuration properties, such as SSL>
```

enabled
Whether this function is enabled. Allowed values: `true`, `false`. This key is optional. Default: `false`.

To enable this function, you must specify `enabled: true`.

Specifying `enabled: false` has the same effect as commenting-out the parent key of this `enabled` key and all descendants of that parent key.

servers
A string containing one or more host/port pairs to use for establishing the initial connection to the Kafka cluster. Use a comma to separate host/port pairs:

```
servers: <host>:<port>
```

or

```
servers: <host1>:<port1>,<host2>:<port2>,...
```

The value of the `servers` key is a string, not a YAML sequence.

topic
Optional Kafka topic name.

If you omit the `topic` key, then OMEGAMON Data Connect sends data for each table to a separate topic.

The per-table topic names have the following pattern:

<topic_prefix>.<product>.<table_name>

where *<topic_prefix>* is the value of the `topic-prefix` key.

Example per-table topic name:

`odp.km5.ascpuutil`

topic-prefix
Optional prefix for per-table Kafka topic names. Default: `odp`.

If you specify a `topic` key, then the `topic-prefix` key is ignored.

filter
Optional filter to restrict what data to send.

This output-level filter applies only to Kafka output, replacing any global-level filter (`connect.filter`).

properties
Optional Kafka producer configuration properties.

For example:

```
"[reconnect.backoff.max.ms]": 30000
```

Important: Enclose Kafka producer property names in square brackets, and then in double quotes.

You can specify *any* Kafka producer configuration properties under this key. For comprehensive details on Kafka producer configuration properties, see the Apache Kafka documentation.

To configure a secure connection between OMEGAMON Data Connect and the Kafka cluster, specify SSL properties. The following listing shows a typical subset of SSL properties:

```
"[security.protocol]": SSL

# Server certificates
"[ssl.truststore.location]": <file_path>
"[ssl.truststore.password]": <string>
"[ssl.truststore.type]": <JKS|PKCS12>

# Client certificate
# (only required if the Kafka server requires client authentication)
"[ssl.keystore.location]": <file_path>
"[ssl.keystore.password]": <string>
"[ssl.keystore.type]": <JKS|PKCS12>
```

SSL using RACF key rings

If you run OMEGAMON Data Connect on z/OS, and you want to use RACF key rings to configure SSL, then you need to use the SSL engine provided with OMEGAMON Data Provider instead of the default SSL engine for Kafka producers. The default SSL engine does not support RACF key rings as a store type.

To use the OMEGAMON Data Provider SSL engine, in addition to setting the `security.protocol` property as you would for the default SSL engine, set the `ssl.engine.factory.class` property:

```
"[security.protocol]": SSL
"[ssl.engine.factory.class]":
com.rocketsoft.odp.server.output.kafka.KafkaSslEngineFactory
```

If you set the `ssl.engine.factory.class` property to the OMEGAMON Data Provider SSL engine, then OMEGAMON Data Connect does not use any of the other `ssl.*` properties.

Instead, the OMEGAMON Data Provider SSL engine has its own set of properties, with the prefix **odp**`.ssl.*`:

```
"[odp.ssl.cipher.suites]": <ciphers_list>
"[odp.ssl.enabled.protocols]": <protocols_list>
"[odp.ssl.protocol]": <protocol>
"[odp.ssl.key.alias]": <string>
"[odp.ssl.keystore.location]": <string>
"[odp.ssl.keystore.password]": <string>
"[odp.ssl.keystore.type]": <JKS|PKCS12|JCERACFKS>
"[odp.ssl.truststore.location]": <string>
"[odp.ssl.truststore.password]": <string>
"[odp.ssl.truststore.type]": <JKS|PKCS12|JCERACFKS>
```

Note: The default SSL engine supports Privacy-Enhanced Mail (PEM) as a store type, but the OMEGAMON Data Provider SSL engine does not. Typically, security providers in the Java runtime environment (JRE) provide support for store types. However, in this case, the default SSL engine contains its own built-in support for PEM. The OMEGAMON Data Provider SSL engine does not contain built-in support for PEM.

odp.ssl.cipher.suites
A list of candidate ciphers for the connection, in one of the following formats:

- OpenSSL cipher list
- A comma-separated list of ciphers using the standard OpenSSL cipher names or the standard JSSE cipher names

This key is optional. Example, in OpenSSL cipher list format:

`HIGH:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!kRSA`

odp.ssl.enabled.protocols
List of protocols to enable.

This key is optional. Example:

`TLSv1.3,TLSv1.2`

odp.ssl.protocol
Protocol to use.

If this protocol is not supported by both ends of the connection, then the connection can fall back (downgrade) to one of the other enabled protocols.

This key is optional. Default in Java 17: `TLSv1.3`.

odp.ssl.key.alias
Alias of the client private key and associated client certificate in the keystore. On z/OS, also known as the certificate *label*.

This key is optional. Default: the default certificate in the keystore.

odp.ssl.keystore.location
Location of the keystore that contains the client certificate.

A keystore is required only if the server requires client authentication.

The location format depends on the keystore type:

JKS
Keystore file path. Example:

`/u/my/security/certs/keystore.jks`

PKCS12
Keystore file path. Example:

`/u/my/security/certs/keystore.p12`

JCERACFKS
Only valid if OMEGAMON Data Connect runs on z/OS.

RACF key ring, in the following format:

`safkeyring://<owner_user_id>/<key_ring_name>`

Note: In this specific context, follow `safkeyring:` with two (2) consecutive slashes.

where *<owner_user_id>* is the RACF user ID that owns the key ring and *<key_ring_name>* is the RACF key ring name.

odp.ssl.keystore.password
Password to access the keystore.

If the keystore type is JCERACFKS, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

odp.ssl.keystore.type
Keystore type. Supported types depend on the security providers in the JRE. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore (*key ring*). Only available if OMEGAMON Data Connect is running on z/OS and the IBMZSecurity provider is available in the JRE.

odp.ssl.truststore.location
Location of the truststore that contains trusted server certificates. See the list of values for `odp.ssl.keystore.location`.

odp.ssl.truststore.password
Password to access the truststore.

If the truststore type is JCERACFKS, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

odp.ssl.truststore.type
Truststore type. See the list of values for `odp.ssl.keystore.type`.

Tip: Conversely, if you specify `odp.ssl.*` properties, but the default SSL engine is selected, then the OMEGAMON Data Connect log contains information messages that report the `odp.ssl.*` properties are "supplied but are not used yet". In any case, the presence of properties for the other SSL engine is benign, and does not affect the behavior of the selected SSL engine.

Example: Per-table topics

The following example sends data to per-table topics in Kafka.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379
```

```
  output:
    kafka:
      enabled: true
      servers: kafka.example.com:9095
```

The topic names match the pattern:

odp.*<product>.<table_name>*

Example: Per-table topics with a custom topic name prefix

The following example sends data to per-table topics in Kafka. The only difference to the previous example is the prefix of the topic names.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka.example.com:9095
      topic-prefix: omegamon
```

In this example, the topic names match the pattern:

omegamon.*<product>.<table_name>*

Example: Send fields from all tables to a single topic

The following example sends all data to a single Kafka topic named omegamon-json.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka.example.com:9095
      topic: omegamon-json
```

Example: Connection with TLS using PKCS12

The following example sends data to Kafka over a secure connection using the default SSL engine.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka1.example.com:9095,kafka2.example.com:9095
      properties:
        "[reconnect.backoff.max.ms]": 30000

        "[security.protocol]": SSL

        # Server certificates
        "[ssl.truststore.location]": /u/my/security/certs/omdp-kafka-
server.p12
```

```
        "[ssl.truststore.password]": Pa$$w0rdTS
        "[ssl.truststore.type]": PKCS12

        # Client certificate
        # (only required if the Kafka server requires client authentication)
        "[ssl.keystore.location]": /u/my/security/certs/omdp-connect.p12
        "[ssl.keystore.password]": Pa$$w0rdKS
        "[ssl.keystore.type]": PKCS12
```

Example: Connection with TLS using a RACF key ring

The following example sends data to Kafka over a secure connection using the OMEGAMON Data Provider SSL engine to refer to a RACF key ring. In this example, the same key ring is used as both a truststore and a keystore.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka1.example.com:9095,kafka2.example.com:9095
      properties:
        "[reconnect.backoff.max.ms]": 30000
        "[security.protocol]": SSL
        "[ssl.engine.factory.class]":
          com.rocketsoft.odp.server.output.kafka.KafkaSslEngineFactory
        "[odp.ssl.truststore.location]": safkeyring://STCOMDP/OMDPring
        "[odp.ssl.truststore.password]": password
        "[odp.ssl.truststore.type]": JCERACFKS
        "[odp.ssl.keystore.location]": safkeyring://STCOMDP/OMDPring
        "[odp.ssl.keystore.password]": password
        "[odp.ssl.keystore.type]": JCERACFKS
```

In this example, the keystore and truststore passwords are the literal string `password`; this is the actual value, not a placeholder.

Example: Connection with TLS using a RACF key ring for the keystore and a PKCS12 file for the truststore

The following example sends data to Kafka over a secure connection using the OMEGAMON Data Provider SSL engine to refer to a RACF key ring for the keystore and a PKCS12 file for the truststore.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka1.example.com:9095,kafka2.example.com:9095
      properties:
        "[security.protocol]": SSL
        "[ssl.engine.factory.class]":
          com.rocketsoft.odp.server.output.kafka.KafkaSslEngineFactory
        "[odp.ssl.truststore.location]": /u/my/security/certs/omdp-kafka-
server.p12
        "[odp.ssl.truststore.password]": Pa$$w0rdTS
        "[odp.ssl.truststore.type]": PKCS12
        "[odp.ssl.keystore.location]": safkeyring://STCOMDP/OMDPring
```

```
        "[odp.ssl.keystore.password]": password
        "[odp.ssl.keystore.type]": JCERACFKS
```

Example: Connection with TLS using a RACF key ring for the truststore

The following example sends data to Kafka over a secure connection using the OMEGAMON Data Provider SSL engine to refer to a RACF key ring for the truststore. In this example, the Kafka servers do not require client authentication, so there is no need for a keystore.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka1.example.com:9095,kafka2.example.com:9095
      properties:
        "[security.protocol]": SSL
        "[ssl.engine.factory.class]":
          com.rocketsoft.odp.server.output.kafka.KafkaSslEngineFactory
        "[odp.ssl.truststore.location]": safkeyring://STCOMDP/OMDPring
        "[odp.ssl.truststore.password]": password
        "[odp.ssl.truststore.type]": JCERACFKS
```

Example: `ssl.*` properties present but unused

The following example has exactly the same effect as the previous example.

```
connect:
  input:
    ...

  output:
    kafka:
      enabled: true
      servers: kafka1.example.com:9095,kafka2.example.com:9095
      properties:
        "[security.protocol]": SSL
        "[ssl.engine.factory.class]":
          com.rocketsoft.odp.server.output.kafka.KafkaSslEngineFactory
        "[odp.ssl.truststore.location]": safkeyring://STCOMDP/OMDPring
        "[odp.ssl.truststore.password]": password
        "[odp.ssl.truststore.type]": JCERACFKS
        # The remaining properties have no effect: the ODP SSL engine does
not use them
        "[ssl.truststore.location]": /u/my/security/certs/omdp-kafka-
server.p12
        "[ssl.truststore.password]": Pa$$w0rdTS
        "[ssl.truststore.type]": PKCS12
        "[ssl.keystore.location]": /u/my/security/certs/omdp-connect.p12
        "[ssl.keystore.password]": Pa$$w0rdKS
        "[ssl.keystore.type]": PKCS12
```

This example demonstrates that, if `ssl.engine.factory.class` selects the OMEGAMON Data Provider SSL engine, then OMEGAMON Data Connect does not use any other `ssl.*` properties.

Related reference

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Characteristics of JSON output from OMEGAMON Data Connect

If you need to work directly with JSON output from OMEGAMON Data Connect, then it's useful to understand the characteristics of this data, such as its structure, property names, and property values.

Prometheus output parameters

OMEGAMON Data Connect can publish attributes to a Prometheus endpoint. OMEGAMON Data Connect Prometheus output parameters describe the Prometheus endpoint and which attributes to publish.

OMEGAMON Data Connect runs an HTTP(S) server that serves the Prometheus endpoint URL. In HTTP(S) terms, OMEGAMON Data Connect is the server and Prometheus is the client.

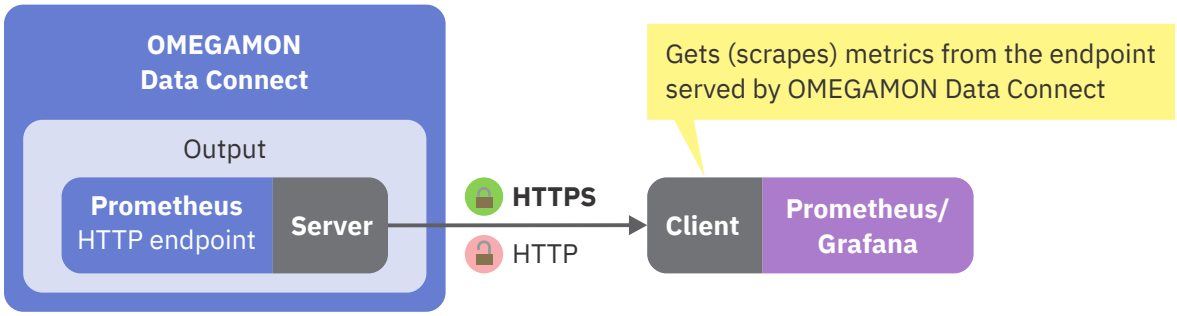


Figure 29. OMEGAMON Data Connect configuration: Prometheus output from an HTTP(S) perspective

In Prometheus architecture, OMEGAMON Data Provider is a *target*. A Prometheus server collect metrics from OMEGAMON Data Provider by scraping metrics from the endpoint served by OMEGAMON Data Connect.

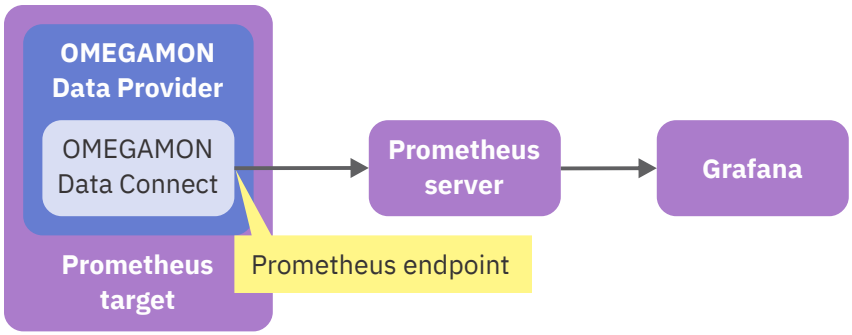


Figure 30. OMEGAMON Data Provider is a Prometheus target

OMEGAMON Data Connect publishes metrics in the Prometheus text-based exposition format. Before specifying Prometheus output parameters for OMEGAMON Data Connect, read the Prometheus documentation for the Prometheus data model and text-based exposition format.

```
connect:
  output:
    prometheus:
      enabled: <true|false> # Default at this level: false
      mappings:
        products:
          <product_code>: # Example: km5, for the z/OS monitoring agent
            enabled: <true|false> # Default at this level: true
            tables:
              <table_name>:
                enabled: <true|false> # Default at this level: true
                metric-expiration: <seconds> # Default depends on the record
                metrics:
                  - <Metrics parameters>
                  - ... # More metrics
                labels:
                  - <field_name>
```

```
                  - ... # More labels
              ...: # More table names
          ...: # More product codes
```

enabled

An `enabled` key can be specified at several levels in the hierarchy of Prometheus output parameters:

- At the highest level, under the `prometheus` key, `enabled` determines whether any metrics are published to Prometheus.

  If you set `enabled` to `false` at this level, then no metrics are published to Prometheus, regardless of parameters at lower levels.

  Default: `false`.

- Under a *<product_code>* key, `enabled` determines whether metrics for that product are published.

  If you set `enabled` to `false` for a product, then no metrics are published for that product, regardless of parameters at lower levels.

  Default: `true`.

- Under a *<table_name>* key, `enabled` determines whether metrics for that table are published.

  Default: `true`.

Allowed values: `true`, `false`.

products

Specifies the product codes of the data sources.

For OMEGAMON monitoring agents as a data source, the product code is the 3-character k*pp* product code of the monitoring agent.

For the product codes of other data sources, see the separate documentation for that software.

tables

Specifies the tables, and metrics from those tables, that OMEGAMON Data Connect publishes to the Prometheus endpoint. The tables and metrics depend on the data source. In the context of OMEGAMON monitoring agents as a data source, the tables are OMEGAMON attribute groups and the metrics are OMEGAMON attributes.

If you omit the `tables` key, then no tables for this product are published.

Each child key of `tables` is a *table_name*. Each *<table_name>* key specifies a list of metrics to publish.

metric-expiration

Optional. Controls when OMEGAMON Data Connect removes metrics from the Prometheus endpoint. Removing metrics from the endpoint is known as *metric expiration*.

The `metric-expiration` parameter specifies a *duration* that OMEGAMON Data Connect uses to calculate the metric expiration time. The metric expiration time is the value of the `write_time` timestamp field in the record being processed plus the duration specified by `metric-expiration`. The metric expiration time is independent of the time that a record arrives at OMEGAMON Data Connect.

Allowed values: a positive integer number of seconds.

The default value depends on whether the `interval_seconds` field is present in the record being processed:

interval_seconds field present?	**Default metric-expiration value**
Yes.	Based on the value of `interval_seconds`.

interval_seconds field present?	**Default metric-expiration value**
For example, records from OMEGAMON monitoring agents.	The following points apply to records from OMEGAMON monitoring agents: • interval_seconds is the collection interval of the historical collection. • write_time is the time that the collection task created the data. • interval_seconds and write_time are fields introduced by OMEGAMON Data Connect, not OMEGAMON attributes. • The collection interval is typically an appropriate basis for Prometheus metric expiration. Hence, for records from OMEGAMON monitoring agents, you typically don't need to set metric-expiration. • Some OMEGAMON monitoring agents collect some attribute groups at fixed intervals. For example, the CICS monitoring agent collects kcpwss attributes every 5 minutes and wss attributes every 15 minutes. If you use attribute groups with fixed collection intervals for Prometheus output, then configure the historical collection interval to match these fixed interval values.
No.	3600 (1 hour).

For information about records from other data sources, such as whether the interval_seconds field is present and what the write_time field represents in those records, see the separate documentation for that software.

Tip: You might want some metrics to always be present when a Prometheus server scrapes the endpoint. For example, if a destination analytics platform calculates aggregated values that rely on a field being present in every record, then gaps in the scraped data can skew results. To avoid gaps, metrics must be updated before they expire: ensure that metric-expiration is greater than the interval between records.

labels
A list of field names in the table to use as metric labels. Typically, labels refer to *string* fields, such as a job, user, or system identifier.

OMEGAMON Data Connect uses labels to map the flat structure of attribute records to the Prometheus dimensional data model.

Metrics parameters

connect.output.prometheus.mappings.products.*<product_code>*.tables.*<table_name>*.metrics:

```
- name: <field_name>
  help: <help_text> # Optional
  type: counter|gauge # Optional (default: gauge)
```

name
Metric name. Must be the field name in the table. Typically, metrics refer to *numeric* fields, such as a timer in seconds or a size in bytes.

In the Prometheus output, OMEGAMON Data Connect prefixes this name with the table name, separated by a underscore.

help
Optional. Metric help text.

type
Optional. Metric type. OMEGAMON Data Connect supports the following Prometheus metric types:

```
counter
gauge
```

Default: `gauge`.

Metrics endpoint URL

The path of the metrics endpoint URL is:

`/metrics`

The hostname and port of the metrics endpoint URL are determined by the Spring Boot server properties `server.address` and `server.port`.

If you do not specify `server.address` or `server.port`, then the default metrics endpoint URL is:

`http://localhost:9070/metrics`

Given the following values:

```
server:
  address: myserver.example.com
  port: 9090
```

the metrics endpoint URL is:

`http://myserver.example.com:9090/metrics`

By default, the endpoint URL uses HTTP, not HTTPS. To use a secure connection (HTTPS), specify `server.ssl` properties.

OMEGAMON attributes versus the Prometheus dimensional data model

OMEGAMON attribute records have a flat structure that consists of a timestamp and a set of attribute key/value pairs.

By contrast, the Prometheus dimensional data model arranges data by metric name and unique combinations of label values.

OMEGAMON Data Connect maps the flat structure of OMEGAMON attributes to the Prometheus dimensional data model based on the labels and metrics that you specify in the configuration parameters, and label values in the incoming attribute data.

Example

Given the following two incoming attribute records (expressed here in JSON format, with line breaks for readability):

```
{
  "write_time": "2021-04-07T05:36:08.773Z",
  "table_name": "cicsrov",
  "cics_region_name": "SCICWEB1", "system_id": "SYSV",
  "cpu_utilization": 10, "transaction_rate": 5
}

{
  "write_time": "2021-04-07T05:36:08.773Z",
  "table_name": "cicsrov",
  "cics_region_name": "SCICWEB2", "system_id": "SYSV",
```

```
  "cpu_utilization": 20, "transaction_rate": 10
}
```

and the following OMEGAMON Data Connect configuration:

```
connect:
  output:
    prometheus:
      enabled: true
      mappings:
        products:
          kc5:
            enabled: true
            tables:
              cicsrov:
                enabled: true
                metrics:
                  - name: transaction_rate
                    type: gauge
                  - name: cpu_utilization
                    type: gauge
                labels:
                  - cics_region_name
                  - system_id
```

then OMEGAMON Data Connect publishes the following data to the Prometheus endpoint:

```
cicsrov_cpu_utilization{cics_region_name="SCICWEB1", system_id="SYSV"} 10
cicsrov_cpu_utilization{cics_region_name="SCICWEB2", system_id="SYSV"} 20
cicsrov_transaction_rate{cics_region_name="SCICWEB1", system_id="SYSV"} 5
cicsrov_transaction_rate{cics_region_name="SCICWEB2", system_id="SYSV"} 10
```

Figure 31. Example Prometheus text-format output

Related reference

Spring Boot server properties
OMEGAMON Data Connect uses the Spring Boot Java framework. The **`server`** key sets Spring Boot server properties.

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Fields introduced by OMEGAMON Data Connect
OMEGAMON Data Connect introduces fields that do not correspond to OMEGAMON attributes.

STDOUT output parameters

OMEGAMON Data Connect STDOUT output parameters specify whether to write data in JSON Lines format to the `stdout` file.

```
connect:
  output:
    stdout:
      enabled: <true|false>
      filter: # Optional output-level filter
        <Filter parameters>
```

`enabled`

Whether this function is enabled. Allowed values: `true`, `false`. This key is optional. Default: `false`.

To enable this function, you must specify `enabled: true`.

Specifying `enabled: false` has the same effect as commenting-out the parent key of this `enabled` key and all descendants of that parent key.

filter
Optional filter to restrict what data to write.

This output-level filter applies only to STDOUT, replacing any global-level filter (`connect.filter`).

Example

```
connect:
  output:
    stdout:
      enabled: true
```

Related reference
Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Characteristics of JSON output from OMEGAMON Data Connect
If you need to work directly with JSON output from OMEGAMON Data Connect, then it's useful to understand the characteristics of this data, such as its structure, property names, and property values.

Filters for JSON-format outputs

You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Note: The filters described here do not apply to the Prometheus output.
The Prometheus output has its own parameters with similar behavior, `connect.output.prometheus.mappings.products.<product_code>.tables`.

You can filter data by product, table, and field name. The product, table, and field names depend on the data source. In the context of OMEGAMON monitoring agents as a data source, the product is an agent, the table is an OMEGAMON attribute group, and the field is an OMEGAMON attribute.

For each table, you can conditionally filter records by specifying an expression. OMEGAMON Data Connect only sends records for which the expression is true.

You can specify a *global-level* filter that applies to all JSON-format outputs and an *output-level* filter for each output. Output-level filters replace any global-level filter.

To specify a global-level filter, insert a `filter` key as a child of the `connect` root key:

```
connect.filter
```

To specify an output-level filter, insert a `filter` key as a child of the key for that output:

```
connect.output.stdout.filter
connect.output.tcp.sinks.<sink_name>.filter
connect.output.http.endpoints.<endpoint_name>.filter
connect.output.kafka.filter
```

If you specify a filter, then only the fields enabled by the filter are sent.

If you do not specify a filter, then all fields from all tables from all products are sent.

You can specify filters inline in the OMEGAMON Data Connect configuration file or in separate *filter include files*.

Global-level and output-level filters have the same format:

```
filter:
  enabled: <true|false>
  include: <file_path> # If specified, the products key is ignored
  products:
    <product_code>:
```

```
      enabled: <true|false>
      tables: # Optional. Default: send all tables from this product
        <table_name>:
          enabled: <true|false>
          condition: # Optional. Default: send all records from this table
            enabled: <true|false>
            expression: <SpEL expression>
            disable-table-on-error: <true|false>
          fields: # Optional. Default: send all fields from this table
            - <field_name>
            - ... # More attribute field names
        ...: # More table names
    ...: # More product codes
```

enabled

An `enabled` key can be specified at several levels in the filter parameters:

- At the highest level, under the `filter` key
- Under a *<product_code>* key
- Under a *<table_name>* key
- Under a `condition` key

Allowed values: `true`, `false`. Default at all levels: `true`.

The `enabled` key has the same effect at every level: setting `enabled: false` is equivalent to omitting, or commenting-out, the parent key and that parent key's descendants.

Key	**Effect of omitting the key, or setting the child key value `enabled: false`**
`filter`	No filter is set. On an output-level filter: causes the global-level filter, if it is enabled, to take effect for that output.
`filter.products.`*<product_code>*	No data from this product is sent.
`filter.products.`*<product_code>*`.tables.`*<table_name>*	No data from this table is sent.
`filter.products.`*<product_code>*`.tables.`*<table_name>*`.` `condition`	No condition is set. All records of this table are sent.

include

Optional. Uses the filter defined in a separate filter include file.

If you specify an `include` key, then OMEGAMON Data Connect ignores the sibling `products` key.

A filter include file is a YAML document that has the same format as the `filter` key in an OMEGAMON Data Connect configuration file, but without the root `filter` key.

Example filter include file:

```
enabled: true
products:
  km5:
    tables:
      ascpuutil: # Send all fields
        enabled: true
```

The filter include *<file_path>* can be absolute or relative. OMEGAMON Data Connect treats a relative file path as being relative to the working directory.

You cannot nest filter includes; you cannot specify an `include` key in a filter include file.

Filter include files must be encoded in UTF-8.

`products`
Only fields from the specified products, such as the specified OMEGAMON monitoring agents, are sent.

Strictly speaking, the `products` key is optional. Omitting the `products` key specifies an "empty" filter with no criteria, which has the same effect as no filter.

If you specify an `include` key, then the `products` key is ignored.

<product_code>
The product code of a data source that you want to filter.

For OMEGAMON monitoring agents as a data source, the product code is the 3-character k*pp* product code of the monitoring agent.

For the product codes of other data sources, see the separate documentation for that software.

You must specify at least one child key under the *<product_code>* key.

To send all tables from the product, omit the child `tables` key and explicitly specify `enabled: true`.

`tables`
Optional. Only fields from the specified tables are sent.

You must specify at least one child *<table_name>* key under the `tables` key.

If all *<table_name>* keys under a `tables` key are set to `enabled: false`, then no data from the product is sent.

<table_name>
The name of a table owned by the product.

You must specify at least one child key under the *<table_name>* key.

To send all fields from the table, omit the `fields` key and explicitly specify `enabled: true`.

`condition`
Optional. OMEGAMON Data Connect only sends records for which the condition expression is true. If the expression is false, OMEGAMON Data Connect discards the record.

The `expression` child key specifies an expression in the Spring Expression Language (SpEL). The expression can test field values in the table. For example:

```
condition:
  expression: cpu_time > 2
```

To test field values, you can either use relational operators or methods:

Expression using a relational operator	Equivalent expression using a method
`syncpoint_elapsed_time == 0`	`syncpoint_elapsed_time.equals(0)`
`cpu_time > 2`	`cpu_time.compareTo(2) > 0`
`cpu_time < 2`	`cpu_time.compareTo(2) < 0`
`(cics_region_name == 'CICSPRD' or cics_region_name == 'TSTRGN1') and syncpoint_elapsed_time == 0`	`cics_region_name.equals('CICSPRD') or cics_region_name.equals('TSTRGN1') and syncpoint_elapsed_time.equals(0)` **Tip:** The `matches` method offers a shorthand for testing alternative values, but uses

Expression using a relational operator	Equivalent expression using a method
	regular expressions, which are typically more computationally expensive: `cics_region_name.matches('CICSPRD|TSTRGN1') and syncpoint_elapsed_time.equals(0)`
`(transaction_id != null) and (transaction_id matches 'PFX.*')`	`transaction_id?.matches('PFX.*')`
`(total_other_wait_times != null) and (total_other_wait_times != 0)`	`total_other_wait_times?.compareTo(0) > 0`

The methods that you can use with a field depend on the Java class to which OMEGAMON Data Connect maps the field: `String`, `Double`, `Integer`, `Long`, or, for timestamp fields such as `write_time`, `OffsetDateTime`. All of these classes support the `equals` and `compareTo` methods. The `String` class also supports the `matches` method, for testing a field value against a regular expression. For details on these and other methods, see the Java documentation for each class.

Attention: Expressions can cause runtime errors or undesirable behavior. Test expressions thoroughly with your data before deploying them in a production environment.

If the expression syntax is invalid, then the Spring framework reports `APPLICATION FAILED TO START`, followed by the error details, and OMEGAMON Data Connect does not start.

If OMEGAMON Data Connect encounters a runtime exception while evaluating the expression, then OMEGAMON Data Connect performs the following actions:

1. Discard the record currently being processed.
2. Report warning message KAYC0057W.
3. Depending on the value of `disable-table-on-error`:

   `false` (default)
   Continue processing records that use the expression.

   `true`

   a. Stop processing records that use the expression; disable the table for outputs that use this filter.

      If the expression is in an output-level filter, then OMEGAMON Data Connect disables the table for that output only. If the expression is in a global-level filter, then OMEGAMON Data Connect disables the table for *all* outputs that use the global-level filter.

   b. Report information message KAYC0056I.

Tip:

- If an expression refers to a field that might not be in every record, then, to avoid throwing a "null pointer" runtime exception, either explicitly test the field for a null value (`<field_name> != null`) or use the safe navigation operator when accessing a method or property of the field. The safe navigation operator is a question mark (?) immediately after the field name.
- Division by integer zero causes an error. However, division by *floating-point* zero does *not* cause an error. For details, see the Java documentation for division by zero.

For more information about SpEL, such as comprehensive details of the operators that you can use in an expression, see the Spring documentation.

To break long expressions over multiple lines in the configuration file, use one of the YAML folding styles. For example, line folding (>-):

```
condition:
  expression: >-
    cics_region_name.matches('CCVQ.*') and
    (total_io_wait_times +
    total_other_wait_times == 0)
```

fields

Optional. A list of field names to send from this table.

OMEGAMON Data Connect always sends the common fields `write_time`, `product_code`, and `table_name`; do not specify these in the list of field names. However, other common fields, such as `interval_seconds`, are sent only if you specify them in this list.

Example: Global-level filter to send all data from one product only

The following filter sends all data from the OMEGAMON z/OS monitoring agent.

```
connect:
  filter:
    products:
      km5: # z/OS
        enabled: true
```

Example: No filter: send all data from all products

The following filter is the same as the previous example except for `enabled: false` directly under the `filter` key, disabling the entire filter.

```
connect:
  filter:
    enabled: false # Disables the entire filter
    products:
      km5:
        enabled: true
```

Example: Global-level filter to send all data from some products only

The following filter sends all data from the OMEGAMON z/OS, CICS, and CICS TG monitoring agents, but blocks all data from other agents. For instance, if OMEGAMON Data Connect receives data from the OMEGAMON Db2 monitoring agent, then OMEGAMON Data Connect does not send the data from that agent, because the filter does not enable the corresponding kd5 product code.

```
connect:
  filter:
    enabled: true
    products:
      km5:
        enabled: true
      kc5: # CICS
        enabled: true
      kgw: # CICS TG
        enabled: true
```

The following filter is equivalent to the previous filter. The resulting behavior is identical. The only difference is that the following filter contains an entry for the Db2 monitoring agent marked `enabled: false` (effectively, a comment).

```
connect:
  filter:
    enabled: true
    products:
      km5:
        enabled: true
      kc5:
        enabled: true
      kgw:
        enabled: true
      kd5: # Db2: do not send
        enabled: false
```

Example: Global-level and output-level filters to send data from different products to different outputs

In the following example:

- The global-level filter sends data from the OMEGAMON z/OS monitoring agent only.
- The Kafka output and the `logstash1` TCP output have no output-level filters, so they use the global-level filter.
- Two of the TCP outputs have output-level filters: the `logstash2` output sends data from the OMEGAMON Db2 and IMS monitoring agents only, and the `splunk` output sends data from the CICS and Java monitoring agents only.
- Only required `enabled` keys are shown; in this example, all of the omitted `enabled` keys default to `true`.

```
connect:
  filter: # Global-level
    products:
      km5:
        enabled: true
  output:
    kafka: # Uses global-level filter
      enabled: true
      servers: kafka.example.com:9095
      topic: omegamon-json
    tcp:
      enabled: true
      sinks:
        logstash1: # Uses global-level filter
          hostname: elastic1.example.com
          port: 5046
        logstash2:
          hostname: elastic2.example.com
          port: 5046
          filter: # Output-level
            products:
              kd5:
                enabled: true
              ki5:
                enabled: true
        splunk:
          hostname: splunk.example.com
          port: 5047
          filter: # Output-level
            products:
              kc5:
```

```
                enabled: true
              kjj:
                enabled: true
```

Example: Filter to send selected fields from one product only

The following global-level filter sends data from the OMEGAMON z/OS monitoring agent: all fields from table `ascpuutil`, but only the specified fields from table `km5wlmclpx`.

```
connect:
  filter:
    enabled: true
    products:
      km5:
        enabled: true
        tables:
          ascpuutil: # Send all fields
            enabled: true
          km5wlmclpx:
            fields: # Send only these fields
              - managed_system
              - class_name
              - class_type
              - transaction_rate
              - transaction_completions
              - transaction_total
```

In the following example, there is no global-level filter. Only the Kafka output is filtered.

```
connect:
  output:
    kafka:
      enabled: true
      servers: kafka.example.com:9095
      topic: omegamon-json
      filter: # Output-level: applies to Kafka output only
        enabled: true
        products:
          km5:
            enabled: true
            tables:
              ascpuutil: # Send all fields
                enabled: true
              km5wlmclpx:
                fields: # Send only these fields
                  - managed_system
                  - class_name
                  - class_type
                  - transaction_rate
                  - transaction_completions
                  - transaction_total
    stdout: # Unfiltered
      enabled: true
    tcp:
      enabled: true
      sinks:
        logstash: # Unfiltered
          hostname: elastic1.example.com
          port: 5046
```

Example: Global-level filter with condition

The following global-level filter restricts output to records of the OMEGAMON z/OS monitoring agent table `ascpuutil` that are for sysplex PLEXA.

```
connect:
  filter:
    enabled: true
    products:
      km5:
        tables:
          ascpuutil: # Send all fields
            condition:
              expression: sysplex_name?.equals('PLEXA')

  output:
    tcp:
      enabled: true
      sinks:
        logstash:
          hostname: elastic1.example.com
          port: 5046
```

Example: Output-level filters with conditions

The following output-level filters send records of the OMEGAMON z/OS monitoring agent table `ascpuutil` to different outputs for different sysplexes.

```
connect:
  output:
    tcp:
      enabled: true
      sinks:
        logstash1: # Sysplex PLEXA output
          hostname: elastic1.example.com
          port: 5046
          filter:
            products:
              km5:
                tables:
                  ascpuutil:
                    condition:
                      expression: sysplex_name?.equals('PLEXA')
        logstash2: # Sysplex PLEXB output
          hostname: elastic2.example.com
          port: 5046
          filter:
            products:
              km5:
                tables:
                  ascpuutil:
                    condition:
                      expression: sysplex_name?.equals('PLEXB')
```

Example: Filter include file

The following TCP output uses a filter include file.

```
connect:
  output:
    tcp:
      enabled: true
      sinks:
```

```
        analytics:
          hostname: analytics.example.com
          port: 5046
          filter:
            include: /var/omdp/filters/analytics.yaml
```

Example: "Empty" output-level filter to send all data from all products

Suppose that you have a global-level filter that restricts output, but you want a particular output to be *unfiltered*. You can achieve this by specifying an output-level filter with `enabled: true` but no criteria; no `products` key or `include` key. The "empty" output-level filter replaces the global-level filter.

In the following example, STDOUT output is unfiltered:

```
connect:
  filter:
    enabled: true
    products:
      km5:
        ascpuutil: # Send all fields
        enabled: true
  output:
    stdout:
      enabled: true
      filter: # Enabled but empty: does not restrict output
        enabled: true
```

Example: Global-level filter that disables the table if a runtime exception occurs in the condition expression

```
connect:
  filter:
    enabled: true
    products:
      km5:
        tables:
          ascpuutil:
            condition:
              expression: sysplex_name.equals('PLEXA') # No safe navigation
operator (?) after sysplex_name
              disable-table-on-error: true # If sysplex_name field is
missing, stop processing records from this table
```

Related reference

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

OMEGAMON attribute dictionary
OMEGAMON Data Connect includes a dictionary of OMEGAMON attributes in a set of YAML files.

TCP output parameters
OMEGAMON Data Connect TCP output parameters specify one or more destinations ("sinks") for sending data in JSON Lines format over a TCP network.

HTTP output parameters
OMEGAMON Data Connect HTTP output parameters specify one or more destinations ("endpoints") for sending data in JSON format in an HTTP/1.1 POST request.

Kafka output parameters
OMEGAMON Data Connect Kafka output parameters specify whether to publish data in JSON format to an Apache Kafka topic.

STDOUT output parameters

OMEGAMON Data Connect STDOUT output parameters specify whether to write data in JSON Lines format to the `stdout` file.

Event publisher parameters

OMEGAMON Data Connect event publisher parameters control aspects of internal OMEGAMON Data Connect processing.

```
connect:
  event-publisher:
    queue-capacity: <number_of_records>
```

`queue-capacity`

The maximum number of records in the queue for each output of OMEGAMON Data Connect.

Default: `50000` (fifty thousand).

Each output of OMEGAMON Data Connect, including each TCP sink and each HTTP endpoint, has its own internal queue.

If a queue reaches capacity:

- The queue rejects (drops) any new incoming records until the queue falls within capacity.
- OMEGAMON Data Connect reports the warning message KAYC0084W.

OMEGAMON Data Connect continues to read incoming records, even if all queues are at capacity.

Queue capacity affects the required maximum heap size (`-Xmx`) of the Java virtual machine that runs OMEGAMON Data Connect.

Attention: If the storage required for all records referred to by all queues exceeds the maximum heap size, OMEGAMON Data Connect shuts down with an out-of-memory condition.

Calculating queue capacity (a number of records) for a given maximum heap size (a number of bytes), or vice versa, is not straightforward:

- The storage, in bytes, depends on the length of each record. Different record types have different lengths.
- The mixture of record lengths can vary over time and depends on site-specific factors such as the OMEGAMON attributes involved and their collection intervals.
- The number of records in each queue can vary over time depending on site-specific factors such as data volume and the processing speed of each destination.

In practice, the default queue capacity of fifty thousand records meets typical requirements, and fits within the maximum heap size of 4096 MB that is set in the supplied sample files for running OMEGAMON Data Connect.

Related reference

Java command line to run OMEGAMON Data Connect
Whichever platform you choose, you can use a Java command line to run OMEGAMON Data Connect.

Connect-specific logging parameters

OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

```
connect:
  logging:
    flood-control:
      enabled: <true|false> # Default: true
      interval: <seconds>
      limit: <records>
```

`flood-control`

Some events can occur frequently, resulting in numerous duplicate log messages. To avoid duplicate messages flooding the log, OMEGAMON Data Connect applies flood control to some messages.

`enabled`

Whether to enable flood control. Allowed values: `true`, `false`. Default: `true`.

Attention: If you disable flood control, then the OMEGAMON Data Connect log could contain *many* duplicate log messages for frequently occurring events.

`interval`

Flood control interval, in seconds. Must be either 0 or a positive integer. Default: 300 (5 minutes).

A positive integer specifies the duration of a *rolling* interval. OMEGAMON Data Connect resets the counter for each flood-controlled message at the end of each interval.

A value of 0 specifies an *indefinite* interval. OMEGAMON Data Connect never resets the flood-controlled message counters. The `limit` applies to the entire duration of the current instance of OMEGAMON Data Connect.

limit

The maximum number of instances of a particular message, including message variable values, allowed with the interval. Must be either 0 or a positive integer. Default: 1 (only one instance of a particular message per interval; no duplicates).

A value of 0 suppresses *all* flood-controlled messages.

To allow duplicate messages within the interval, increase the limit.

OMEGAMON Data Connect applies flood control to the following messages:

KAYC0031W
KAYC0057W
KAYC0061W
KAYC0062W
KAYC0079E
KAYC0084W

At the end of each rolling interval, if any messages have been suppressed, OMEGAMON Data Connect reports message KAYC0074I, followed by a report of the suppressed messages and the number of messages suppressed.

Example: Flood control interval of 10 minutes, with a limit of 10 instances of the same message within each interval

```
connect:
  logging:
    flood-control:
      enabled: true
      interval: 600
      limit: 10
```

Example: Indefinite flood control interval, with a limit of 10 instances of the same message for the duration of the current instance of OMEGAMON Data Connect

```
connect:
  logging:
    flood-control:
      enabled: true
      interval: 0 # Indefinite, not rolling
      limit: 10
```

Example: Disable flood control

```
connect:
  logging:
    flood-control:
      enabled: false
```

Related reference

Spring Boot logging properties
OMEGAMON Data Connect uses the Spring Boot Java framework. The **`logging`** key sets Spring Boot logging properties.

Related information

KAYC0031W
Event publication error

KAYC0057W
Filter condition for *product_code.table_name* failed, expression '*expression*'

KAYC0061W
Malformed record received

KAYC0062W
Mapping class not found for *product_code.table_name table_version*

KAYC0074I
total_messages_suppressed messages have been suppressed in *logger_name* in the last *flood_control_interval* seconds:

Spring Boot server properties

OMEGAMON Data Connect uses the Spring Boot Java framework. The **`server`** key sets Spring Boot server properties.

In the context of publishing Prometheus or actuator endpoints over HTTPS, OMEGAMON Data Connect is the *server*.

To publish Prometheus output and Spring Boot Actuator endpoints, OMEGAMON Data Connect uses the Spring Boot server address, port, and SSL properties.

Only some Spring Boot server properties are described here. For more details on these and other Spring Boot server properties, see the Spring Boot documentation.

```
server:
  address: <string>
  port: <number>
  ssl: # Required only for HTTPS, not HTTP
    <SSL properties>
```

`address`
Hostname or IP address on which to listen for requests. Default: `localhost`.

`port`
Port number on which to listen for requests. Default: `9070`.

SSL properties

SSL properties are required only if you want to use HTTPS rather than HTTP.

Transport Layer Security (TLS) supersedes the deprecated Secure Sockets Layer (SSL) protocol. However, for historical reasons, the term SSL is sometimes still used when not referring to a specific protocol.

Only some Spring Boot server SSL properties and allowed values are described here. For example, this documentation does not describe all types of keystore and truststore. For more details on these and other Spring Boot server SSL properties and allowed values, see the Spring Boot documentation.

```
enabled: <true|false>
ciphers: <ciphers_list>
client-auth: <need|none|want>
enabled-protocols: <protocols_list>
protocol: <protocol>
key-alias: <string>
key-password: <string>
key-store: <string>
key-store-password: <string>
key-store-type: <JKS|PKCS12|JCERACFKS>
trust-store: <string>
trust-store-password: <string>
trust-store-type: <JKS|PKCS12|JCERACFKS>
```

enabled
Whether to enable SSL/TLS:

true
Use HTTPS.

false
Use HTTP.

This key is optional. Default: `true`.

Use `enabled: false` as a convenient single-line method for falling back to HTTP, as an alternative to using YAML comment syntax to comment-out all of the SSL properties.

ciphers
A list of candidate ciphers for the connection, in one of the following formats:

- OpenSSL cipher list
- A comma-separated list of ciphers using the standard OpenSSL cipher names or the standard JSSE cipher names

This key is optional. Example, in OpenSSL cipher list format:

```
HIGH:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!kRSA
```

client-auth
Client authentication. Whether to request a client certificate from the client, and then whether to allow the connection based on the client response.

need
Request a client certificate. Allow the connection only if the client responds with a valid certificate.

none
Do not request a client certificate. Allow the connect without client authentication.

want
Request a client certificate. If the client responds with a certificate, allow the connection only if the certificate is valid. If the client does not respond with a certificate, allow the connection.

enabled-protocols
List of protocols to enable.

This key is optional. Example:

```
TLSv1.3,TLSv1.2
```

protocol
Protocol to use.

If this protocol is not supported by both ends of the connection, then the connection can fall back (downgrade) to one of the other enabled protocols.

This key is optional. Default in Java 17: `TLSv1.3`.

key-alias
Alias of the server private key and associated server certificate in the keystore. On z/OS, the alias is also known as the certificate *label*.

This key is optional. Default: the default certificate in the keystore.

key-password
Password required to access the server private key in the keystore.

This key is optional. Default: the value of `key-store-password`.

key-store
Location of the keystore that contains the server certificate.

The location format depends on the keystore type:

JKS
Keystore file path. Example:

```
/path/to/keystore.jks
```

PKCS12
Keystore file path. Example:

```
/path/to/keystore.p12
```

JCERACFKS
Only valid if OMEGAMON Data Connect runs on z/OS.

RACF key ring, in the following format:

`safkeyring://`*`<owner_user_id>`*`/`*`<key_ring_name>`*

Note: In this specific context, follow `safkeyring:` with two (2) consecutive slashes.

where *<owner_user_id>* is the RACF user ID that owns the key ring and *<key_ring_name>* is the RACF key ring name.

key-store-password
Password to access the keystore.

If the keystore type is JCERACFKS, then specify the fixed value:

```
password
```

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

key-store-type
Keystore type. Supported types depend on the security providers in the JRE. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore (*key ring*). Only available if OMEGAMON Data Connect is running on z/OS and the IBMZSecurity provider is available in the JRE.

trust-store
Location of the truststore that contains trusted client certificates. See the list of example locations for `key-store`.

A truststore is required only for client authentication; that is, when the value of `client-auth` is `need` or `want`.

`trust-store-password`
Password to access the truststore.

If the truststore type is `JCERACFKS`, then specify the fixed value:

`password`

RACF does not use this value for authentication; this value is required only for compatibility with the JCE requirement for a password.

`trust-store-type`
Truststore type. See the list of example types for `key-store-type`.

Example: HTTPS with client authentication, using the same RACF key ring as both keystore and truststore

```
server:
  address: 0.0.0.0
  port: 9080
  ssl:
    enabled: true
    enabled-protocols: TLSv1.2
    protocol: TLS
    client-auth: need
    # Server certificate
    key-store: safkeyring://STCOMDP/OMDPring
    key-store-type: JCERACFKS
    key-store-password: password # Required fixed value
    key-alias: OMDPcert
    # Trusted client certificates
    trust-store: safkeyring://STCOMDP/OMDPring
    trust-store-type: JCERACFKS
    trust-store-password: password # Required fixed value
```

Example: HTTPS with client authentication, using JKS keystore and PKCS12 truststore

```
server:
  address: 0.0.0.0
  port: 9080
  ssl:
    enabled: true
    enabled-protocols: TLSv1.2
    protocol: TLS
    client-auth: need
    # Server certificate
    key-store: /u/my/security/keystore.jks
    key-store-type: JKS
    key-store-password: pa$$w0rdKS
    key-alias: OMDPcert
    # Trusted client certificates
    trust-store: /u/my/security/truststore.p12
    trust-store-type: PKCS12
    trust-store-password: pa$$w0rdTS
```

Related reference
Prometheus output parameters

OMEGAMON Data Connect can publish attributes to a Prometheus endpoint. OMEGAMON Data Connect Prometheus output parameters describe the Prometheus endpoint and which attributes to publish.

Spring Boot logging properties

OMEGAMON Data Connect uses the Spring Boot Java framework. The **`logging`** key sets Spring Boot logging properties.

To control the logging level for OMEGAMON Data Connect, set the `logging.level.com.rocketsoft` property value:

```
logging:
  level:
    com:
      rocketsoft: <level>
```

Allowed values for *<level>*, from lowest to highest logging level: `ERROR` (lowest), `WARN`, `INFO` (default), `DEBUG`, `TRACE` (highest).

For details of logging levels and their meanings, see the Spring Boot documentation.

Example: Set the logging level in the configuration file

Set the `logging.level.com.rocketsoft` property value in the YAML configuration file, `connect.yaml`.

```
logging:
  level:
    com:
      rocketsoft: INFO
```

Example: Set the logging level in the z/OS JCL procedure

If you use the supplied sample `KAYCONN` JCL procedure to run OMEGAMON Data Connect, you can set the logging level by setting the value of the `LOGLEVEL` symbolic parameter in the procedure JCL:

```
// SET LOGLEVEL='ERROR'
```

Related reference

Connect-specific logging parameters
OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

Configuration validator

The OMEGAMON Data Provider configuration validator is a command-line tool that you can use to validate the OMEGAMON Data Provider configuration files that are in YAML format: the OMEGAMON Data Connect configuration file and, if you are using OMEGAMON monitoring agents as a data source, the OMEGAMON Data Provider collection configuration member, `RKANPARU(KAYOPEN)`.

Whenever you edit one of these configuration files, use the configuration validator as a "preflight" check before using the edited file to configure your system.

The configuration validator is a Java application. To run the configuration validator on operating systems such as z/OS UNIX and Linux, use the supplied shell script.

Note: OMEGAMON Data Provider does not supply a corresponding "preflight " tool to validate OMEGAMON Data Broker configuration parameters.

Prerequisites

The configuration validator requires Java 17 or later, 64-bit edition.

Location

The configuration validator is supplied in the OMEGAMON Data Connect installation directory.

More specifically, the shell script to run the configuration validator is supplied under the OMEGAMON Data Connect installation directory in the relative file path:

`bin/validate`

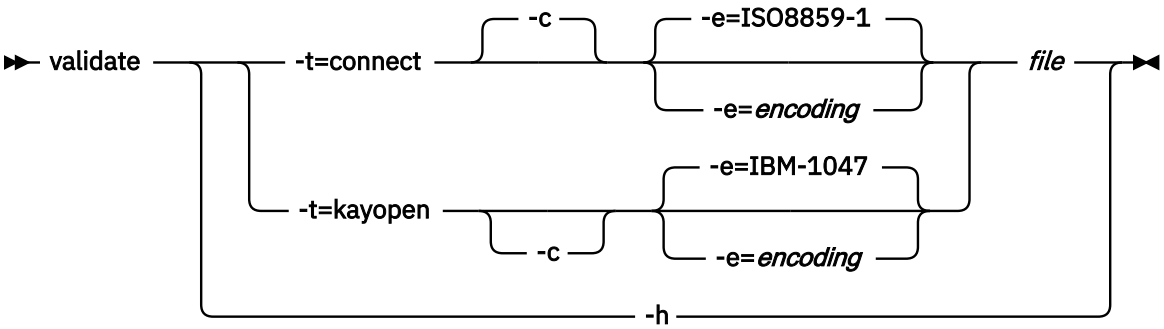
The script file name `validate` has no file extension (no trailing `.sh`).

Example absolute file path:

`/usr/lpp/omdp/kay-110/bin/validate`

Tip: Run the script from its supplied location in the installation directory. Don't copy the script to a different directory and try to run it from there. The script uses its own path to locate files in the installation directory. If you try to run the script from a different directory, the script won't find those files. If you want to make the script available in a directory that is already listed in your PATH environment variable, then, in that other directory, create a symbolic link that refers to the script in the installation directory. Running the symbolic link will run the script in the installation directory.

Syntax



Options

`-t=connect|kayopen`
The type of file to be validated:

`connect`
OMEGAMON Data Connect configuration file

`kayopen`
OMEGAMON Data Provider collection configuration member

`-c, --case-sensitive`
Sets the validation to be case-sensitive.

The default validation case-sensitivity depends on the type of file to be validated:

- When `-t=connect`, validation is case-sensitive by default. Validation of OMEGAMON Data Connect configuration files is *always* case-sensitive.
- When `-t=kayopen`, validation is case-insensitive by default. If you set validation of the KAYOPEN member to be case-sensitive, then the following items must be all lowercase:
  - Keys (parameter names).
  - Values that are constrained to an enumerated list. For example, values of the `product` key.

-e, --encoding=*encoding*
Overrides the default character encoding that the configuration validator uses to interpret the file to be validated.

The default encoding depends on the type of file to be validated:

- When `-t=connect`, the default encoding is the EBCDIC code page IBM-1047.
- When `-t=kayopen`, the default encoding is the extended ASCII code page ISO8859-1.

file
The configuration file to be validated.

To refer to a file in a z/OS UNIX or Linux file system, specify the file path. For example:

```
/var/omdp/prod-a/config/connect.yaml
```

To refer to a z/OS MVS data set name, begin with a double quotation mark, followed by two slashes, then the fully qualified data set name enclosed in single quotation marks, and then end with another double quotation mark. For example:

```
"//'OMEGHLQ.RTENAME.RKANPARU(KAYOPEN)'"
```

-h, --help
Display help information and then exit.

Usage

After editing a configuration file, run the configuration validator shell script. See the examples at the end of this topic.

If the configuration file is valid, the configuration validator reports the following message:

```
This configuration file is valid.
```

Otherwise, the configuration validator reports the following message:

```
Validation failed.
```

followed by one or messages describing validation errors.

Limitations

The configuration validator is a useful preflight tool that catches many issues that can cause errors at run time. However, there are limitations to this validation. Issues not caught by the configuration validator can still cause errors at run time.

When the validator reports that a configuration file is *valid*, it means that the file meets the following two conditions:

- The configuration file is a well-formed YAML document. The configuration file follows YAML syntax rules, independent of any rules specific to OMEGAMON Data Provider.
- The configuration file follows the limited set of rules, specific to OMEGAMON Data Provider, such as required keys and values, defined in a schema that is built into the validator.

The following limitation applies to validating any configuration file:

Most values are checked for data type, some values are checked for constraints
For most keys in the configuration file, the validator checks that the value matches the expected data type. For example, if you specify a string value for a key that expects an integer, the validator reports `string found, integer expected`.

For some keys, the validator also performs more specific constraint checks. For example:

- The validator checks whether the value of the `product` key in the KAYOPEN member matches the product code of a supported OMEGAMON monitoring agent.

- The validator does not check whether the value of the `product` key in the OMEGAMON Data Connect configuration file matches the product code of a supported OMEGAMON monitoring agent. In the context of OMEGAMON Data Connect, the `product` key can refer to other data sources, not just OMEGAMON monitoring agents.

The following limitations apply only to validating OMEGAMON Data Connect configuration files:

Only some Spring Boot server properties checked
The root-level **`server`** key sets Spring Boot server properties. The validator only checks the few Spring Boot server properties that are described in this OMEGAMON Data Provider documentation. The validator does not check the many other server properties that you can specify.

Spring Boot logging properties not checked
Descendants of `logging.level` are not checked.

Kafka producer properties not checked
Descendants of `connect.output.kafka.properties` are not checked.

Filter include files not checked
OMEGAMON Data Connect configuration files can refer to filters that are defined in separate filter include files. The validator checks neither the contents of those files nor whether those files exist.

To validate the contents of a filter include file, you can insert the contents under a `filter` key in a minimal configuration file. For example:

```
connect:
  input:
    tcp:
      hostname: none
      port: 1
  output:
    stdout:
      filter:
        # Insert contents of filter include file here
```

When you insert the contents of the filter include file, remember to indent it under the `filter` key.

Example: Validating an OMEGAMON Data Connect configuration file

At a shell prompt, change to the `bin` directory under the OMEGAMON Data Connect installation directory, and then enter the following command:

```
./validate -t=connect /var/omdp/prod-a/config/connect.yaml
```

Example: Validating RKANPARU(KAYOPEN)

At a shell prompt, change to the `bin` directory under the OMEGAMON Data Connect installation directory, and then enter the following command:

```
./validate -t=kayopen "//'OMEGHLQ.RTENAME.RKANPARU(KAYOPEN)'"
```

Example: Incorrect indentation invalidates a configuration file

Given the following OMEGAMON Data Connect configuration file:

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379
  output:
```

```
    stdout:
    enabled: true # Incorrectly indented
```

the configuration validator reports the following messages:

```
Validation failed.
$.connect.output.stdout: null found, object expected
$.connect.output.enabled: is not defined in the schema and the schema does not
allow additional properties
```

Here, the message phrase "not defined in the schema" means, more specifically, "not defined in the schema *at this indent level*".

The `enabled` key is unknown at the same indent level as `stdout`. Instead, `enabled` should be indented as a child of `stdout`.

To make this configuration file valid, you need to insert spaces to indent `enabled` under `stdout`.

Related information

OMEGAMON Data Connect fails due to unknown configuration parameters

Characteristics of output from OMEGAMON monitoring agents

The general characteristics of output from OMEGAMON Data Provider depend on the OMEGAMON Data Connect configuration, and are independent of the data source. However, some specific characteristics depend on the data source.

For example:

- The output data format, such as JSON, depends on the OMEGAMON Data Connect configuration.
- Property names in the output, such as JSON keys, depend on the data source.

This OMEGAMON Data Provider documentation describes characteristics of output where OMEGAMON monitoring agents are the data source.

For characteristics of output for other data sources, see the separate documentation for that software.

OMEGAMON monitoring agents supported by OMEGAMON Data Provider

OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Table 6. OMEGAMON monitoring agents supported by OMEGAMON Data Provider, with links to attributes documentation

Product code	Monitored subsystem	Monitoring agent	Minimum version of monitoring agent supported
kc5	CICS	IBM Z OMEGAMON for CICS Previous name, before version 5.6: IBM Z OMEGAMON for CICS on z/OS	5.5
kgw	CICS TG	IBM Z OMEGAMON for CICS TG Previous name, before version 5.6: IBM Z OMEGAMON for CICS TG on z/OS	5.5
kd5	Db2	IBM OMEGAMON for Db2 Performance Expert on z/OS Previous name, before version 5.5: IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS	5.4
ki5	IMS	IBM OMEGAMON for IMS on z/OS	5.5
kjj	JVM	IBM Z OMEGAMON AI for JVM Previous name, before version 6.1: IBM Z OMEGAMON for JVM	5.5
kmq	MQ	IBM OMEGAMON for Messaging on z/OS	7.5

Table 6. OMEGAMON monitoring agents supported by OMEGAMON Data Provider, with links to attributes documentation (continued)

Product code	Monitored subsystem	Monitoring agent	Minimum version of monitoring agent supported
kn3	Network	IBM Z OMEGAMON AI for Networks Previous name, before version 6.1: IBM Z OMEGAMON Network Monitor	5.6
ks3	Storage	IBM OMEGAMON for Storage on z/OS	5.5
km5	z/OS	IBM Z OMEGAMON AI for z/OS Previous name, before version 6.1: IBM Z OMEGAMON Monitor for z/OS	5.6

Some versions of some supported monitoring agents *contain* OMEGAMON Data Provider. For details, see "A product offering that contains OMEGAMON Data Provider" on page 44.

A similar mapping of product codes to monitoring agent product names is available in YAML format in the attribute dictionary included with OMEGAMON Data Connect.

OMEGAMON Data Provider is designed to be extended to support more agents.

Product code

Each agent has a unique k*pp* product code. The product code matches the agent configuration parameter prefix.

You use product codes to configure the behavior of OMEGAMON Data Provider:

- OMEGAMON Data Provider collection: which collections to send to OMEGAMON Data Connect
- OMEGAMON Data Connect filters: which data to send to each output

Output from OMEGAMON Data Provider contains the product code in the `product_code` common field.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member `RKANPARU(KAYOPEN)` configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

Prometheus output parameters
OMEGAMON Data Connect can publish attributes to a Prometheus endpoint. OMEGAMON Data Connect Prometheus output parameters describe the Prometheus endpoint and which attributes to publish.

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Fields introduced by OMEGAMON Data Connect
OMEGAMON Data Connect introduces fields that do not correspond to OMEGAMON attributes.

Attribute groups versus table names

OMEGAMON Data Provider uses concise table names to refer to attribute groups.

OMEGAMON attribute dictionary

OMEGAMON Data Connect includes a dictionary of OMEGAMON attributes in a set of YAML files.

The dictionary describes OMEGAMON monitoring agent product codes, table names, and attribute field names.

The dictionary is a useful reference for various tasks. For example:

- Configuring OMEGAMON collection tasks to select collections to send to OMEGAMON Data Broker based on product code and table name.
- Configuring OMEGAMON Data Connect to filter attributes based on product code, table name, and attribute field name.
- Understanding attribute values in output from OMEGAMON Data Provider.

You can use the dictionary files as a human-readable reference or develop programs to parse their contents.

Location of the dictionary files

The dictionary files are supplied in the `dictionary` directory under the OMEGAMON Data Connect installation directory.

Default z/OS UNIX directory path:

`/usr/lpp/omdp/kay-110/dictionary`

Index of supported monitoring agents

The following file contains an index of the monitoring agents supported by OMEGAMON Data Provider:

`dictionary/_index.yaml`

This file maps k*pp* product codes to product names (titles).

Example snippet:

```
products:
  - code: km5
    title: IBM Z OMEGAMON Monitor for z/OS
```

Indexes of tables owned by each monitoring agent

The following files contain indexes of tables owned by each monitoring agent:

`dictionary/k*pp*/_index.yaml`

These files map the concise table names used by OMEGAMON Data Provider to the attribute group names presented in OMEGAMON user interfaces and documentation.

Example snippet of `km5/_index.yaml`:

```
tables:
  - name: ascpuutil
    title: Address Space CPU Utilization
```

Note: OMEGAMON Data Provider supports only the attribute groups that can be included in OMEGAMON historical data collection.

Attributes in each table

The following files describe the attributes in each table:

`dictionary/k`*`pp`*`/`*`table_name`*`.yaml`

Note: There is one exception to the *`table_name`*`.yaml` file naming pattern. The CICS monitoring agent has an attribute group named CICSplex Connection Analysis that has the table name `con`. The Windows operating system reserves the word `con` to refer to the console device and does not allow `con.yaml` as a file name. The attribute dictionary file for the `con` table is `kc5/cont.yaml`, with a `t` appended to the table name.

These files map the snake_case field names used by OMEGAMON Data Provider to the attribute names (titles) presented in OMEGAMON user interfaces and documentation. For example, `job_name` maps to Job Name.

These files also contain a multi-line, plain-text description of each attribute. These descriptions are similar to the attribute descriptions in the monitoring agent documentation.

Example snippet of `km5/ascpuutil.yaml`:

```
name: ascpuutil
title: Address Space CPU Utilization
fields:
  - name: managed_system
    title: Managed System
    description:
      - A z/OS operating system in your enterprise that is being monitored
      - by an IBM Z OMEGAMON Monitor for z/OS agent. Valid value is a
      - character string with a maximum length of 32 bytes.
  - name: job_name
    title: Job Name
    description:
      - The name of the job, started task, TSO user, APPC address space,
      - and so on, consuming CPU cycles. Valid value is a string, with a
      - maximum of eight characters.
```

Related reference

OMEGAMON Data Connect installation directory
The OMEGAMON Data Connect installation directory contains the files for OMEGAMON Data Connect that are supplied with OMEGAMON Data Provider.

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Attribute names versus field names

OMEGAMON attribute names are either not ideal or not usable as field names in some analytics platforms and data formats. OMEGAMON Data Provider converts OMEGAMON attribute names into "safe" field names.

In this context, the term *field name* corresponds to platform- or format-specific terms such as *key*, *property name*, and *metric name*.

Example OMEGAMON attribute name: "MVS Busy%".

Format of OMEGAMON attribute names	Example rules for field names
Several words separated by spaces.	No spaces allowed.
Can contain various non-alphanumeric characters, such as a percent sign (%).	Only a limited subset of non-alphanumeric characters allowed.
Can begin with a digit.	Must begin with a letter.
Mixed-case.	Case-sensitive. Referring to a field name with a single character in the incorrect case can result in an error such as "Field not found".

OMEGAMON Data Provider field names

OMEGAMON Data Provider field names have the following format:

- Snake case, containing only lowercase letters (a - z), digits (0 - 9), and underscores (_).
- Begin with a letter.
- End with a letter or a number; no trailing underscores.

Converting attribute names to field names

OMEGAMON Data Provider field names are the OMEGAMON attribute names after applying the following conversion steps:

1. Lowercase all letters.
2. Replace space, hyphen (-), backslash (\) with underscore (_).
3. Replace slash (/) with underscore (_).

   Exception: replace "I/O" with "io", not "i_o".
4. Replace percent sign (%) with the string "pct".

   Insert a leading or trailing underscore before or after "pct", to separate it from adjacent text, unless that underscore already exists.
5. If the first character is a digit (1, 2, 3, ...), replace it with the corresponding English word (one, two, three, ...).
6. Convert double underscores (__) to a single underscore (_).

Note:

- For a comprehensive mapping of field names to attribute names, see the attribute dictionary included with OMEGAMON Data Connect.
- Each OMEGAMON product documents the attributes that it collects. For example, for attributes collected by IBM Z OMEGAMON AI for z/OS, 5.6, see the corresponding Attributes documentation.
- The OMEGAMON enhanced 3270 user interface (e3270UI) menu option **Tools** > **ODI (Object Definitions)** lists attribute tables and their attributes.

Examples

OMEGAMON attribute name	OMEGAMON Data Provider field name
Buffer Size	`buffer_size`
SCM Service Time	`scm_service_time`
Amount CSA In Use	`amount_csa_in_use`
Total 4K Reqs Completed/Sec	`total_4k_reqs_completed_sec`

OMEGAMON attribute name	OMEGAMON Data Provider field name
Active I/O Threshold	`active_io_threshold`
MVS Busy%	`mvs_busy_pct`
Physical % zIIP	`physical_pct_ziip`
Dependent Enclave IFA % On CP	`dependent_enclave_ifa_pct_on_cp`
CPU % \ MVS Normalized	`cpu_pct_mvs_normalized`
1 Megabyte Writes Demoted	`one_megabyte_writes_demoted`
3rd Device Wait Percentage	`third_device_wait_percentage`

- In JSON output from OMEGAMON Data Provider, the attribute name "Buffer Size" is represented as the key `buffer_size`.
- In a Prometheus endpoint published by OMEGAMON Data Provider, the attribute name "MVS Busy%" is represented as the metric `km5thrsuml_mvs_busy_pct`, where `km5thrsuml` is the table name.

Attribute groups versus table names

OMEGAMON Data Provider uses concise table names to refer to attribute groups.

A mapping of table names to attribute groups is available from several sources:

- YAML-format attribute dictionary supplied with OMEGAMON Data Connect.
- OMEGAMON enhanced 3270 user interface (e3270UI) menu option **Tools** > **ODI (Object Definitions)**.
- Documentation for some monitoring agents. For example:

Monitoring agent	Mapping documentation
IBM Z OMEGAMON AI for z/OS	Historical data table names and corresponding attribute groups
IBM Z OMEGAMON for CICS	OMEGAMON for CICS on z/OS near-term history tables
IBM Z OMEGAMON for CICS TG	OMEGAMON for CICS TG on z/OS near-term history tables

Examples

OMEGAMON Data Provider uses the table name `ascpuutil` to refer to the attribute group "Address Space CPU Utilization".

In the collection configuration member, `RKANPARU(KAYOPEN)`:

```
  - product: km5
    table: ascpuutil
    interval: 0
```

In the OMEGAMON Data Connect configuration file, `config/connect.yaml`:

```
  filter:
    products:
      km5:
        ascpuutil:
          fields:
            - cpu_percent
            - ... # Other field names
```

In JSON output from OMEGAMON Data Provider:

```
"table_name":"ascpuutil"
```

In a Prometheus metric published by OMEGAMON Data Provider:

```
ascpuutil_cpu_percent{labels} value
```

Related reference

OMEGAMON monitoring agents supported by OMEGAMON Data Provider

OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Fields introduced by OMEGAMON Data Connect

OMEGAMON Data Connect introduces fields that do not correspond to OMEGAMON attributes.

Common fields

OMEGAMON Data Connect introduces the following common field to data from all OMEGAMON monitoring agents:

`interval_seconds`
The collection interval of the historical collection, in seconds.

OMEGAMON Data Connect introduces the following common fields to data from *all data sources*, not just data from OMEGAMON monitoring agents:

`product_code`
The 3-character k*pp* product code of the monitoring agent that owns the table.

`table_name`
The concise table name corresponding to the longer, multi-word attribute group name typically presented in OMEGAMON product documentation and user interfaces.

`table_version`
An integer representing the table schema version.

`write_time`
Timestamp when the data was created on z/OS by the OMEGAMON collection task, before it was forwarded to OMEGAMON Data Broker.

For descriptions of the values of the common fields `product_code`, `table_name`, and `write_time` for other data sources, see the separate documentation for that software.

km5: z/OS monitoring agent

OMEGAMON Data Connect introduces the following fields to data from IBM Z OMEGAMON AI for z/OS:

`smf_id`
The SMF ID of the z/OS LPAR from which these attributes were collected.

The `smf_id` field is included only for tables that contain LPAR-specific attributes. If the table contains sysplex-wide attributes, then there is no `smf_id` field.

If `smf_id` already exists as an attribute in a table, then OMEGAMON Data Connect does nothing: the output contains the original field value.

`sysplex_name`
The z/OS sysplex from which these attributes were collected.

If `sysplex_name` already exists as an attribute in a table, then OMEGAMON Data Connect does nothing: the output contains the original field value.

kd5: Db2 monitoring agent

OMEGAMON Data Connect introduces the following fields to data from IBM OMEGAMON for Db2 Performance Expert on z/OS:

db2_subsystem
The Db2 subsystem ID from which these attributes were collected, derived from the originnode attribute value.

mvs_system
The MVS ID of the z/OS LPAR from which these attributes were collected, derived from the originnode attribute value.

ks3: Storage monitoring agent

OMEGAMON Data Connect introduces the following field to data from IBM OMEGAMON for Storage on z/OS:

smf_id
The SMF ID of the z/OS LPAR from which these attributes were collected.

If smf_id already exists as an attribute in a table, then OMEGAMON Data Connect does nothing: the output contains the original field value.

Related reference
OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Characteristics of JSON output from OMEGAMON Data Connect

If you need to work directly with JSON output from OMEGAMON Data Connect, then it's useful to understand the characteristics of this data, such as its structure, property names, and property values.

The characteristics described here refer to data with OMEGAMON monitoring agents as the data source. For characteristics of data from other data sources, see the separate documentation for that other software.

Flat: no nested objects

Each line of the JSON Lines output by OMEGAMON Data Connect is a JSON object consisting of a collection of name/value pairs ("properties").

The structure is flat: there are no nested objects.

No null values

If there is no underlying data available for an OMEGAMON attribute, then rather than representing the attribute in JSON output as a key with the JavaScript value null, OMEGAMON Data Connect omits the key.

OMEGAMON Data Connect performs this processing for each line of JSON. Depending on the availability of the underlying data, a key that is present in some lines of JSON output might not be present in other lines for the same attribute table.

No whitespace between tokens

The JSON standard (ECMA-404) allows insignificant whitespace before or after any token.

The JSON output by OMEGAMON Data Connect is deliberately compact and omits such whitespace.

Property names

JSON property names are based on OMEGAMON attribute names. For details, see “Attribute names versus field names” on page 170.

Timestamps

Timestamps are in ISO 8601 date and time of day representation extended format with a trailing zone designator:

*yyyy-mm-dd*T*hh*:*mm*:*ss*.*SSSSSSSSS*[+|-]*hh*:*mm*

Example:

2021-06-23T00:18:28.999999001-04:00

Scientific notation

Very large or very small numbers might be represented in scientific notation. For example, 1.077952576E8.

Introduced fields

OMEGAMON Data Connect introduces fields that do not occur in the original OMEGAMON attribute groups.

Example

Here is a single line of JSON output from OMEGAMON Data Connect, shown here with indenting and line breaks for readability:

```
{
  "managed_system":"ZOSAPLEX:ZOS1:MVSSYS",
  "job_name":"M5M5DS",
  "cpu_percent":1.7,
  "tcb_percent":1.7,
  "srb_percent":0.0,
  "step_name":"M5M5DS",
  "proc_step":"TEMSREMT",
  "svcclass":"STCLO",
  "svcclasp":1,
  "asid":417,
  "jesjobid":"S0852831",
  "job_cpu_time":1671.63,
  "job_tcb_time":1655.76,
  "job_srb_time":15.34,
  "sysplex_name":"ZOSAPLEX",
  "smf_id":"ZOS1",
  "table_name":"ascpuutil",
  "write_time":"2021-10-13T08:00:13.999999001-04:00",
  "product_code":"km5",
  "interval_seconds":60
}
```

This example includes the following fields introduced by OMEGAMON Data Connect:

```
sysplex_name
smf_id
table_name
write_time
product_code
interval_seconds
```

Related reference

Fields introduced by OMEGAMON Data Connect
OMEGAMON Data Connect introduces fields that do not correspond to OMEGAMON attributes.

TCP output parameters
OMEGAMON Data Connect TCP output parameters specify one or more destinations ("sinks") for sending data in JSON Lines format over a TCP network.

Kafka output parameters
OMEGAMON Data Connect Kafka output parameters specify whether to publish data in JSON format to an Apache Kafka topic.

STDOUT output parameters
OMEGAMON Data Connect STDOUT output parameters specify whether to write data in JSON Lines format to the `stdout` file.

Troubleshooting

Follow these steps to diagnose and correct problems that you experience with OMEGAMON Data Provider. For example, expected data not arriving at a destination analytics platform.

Procedure

1. Examine the messages from the components involved in OMEGAMON Data Provider.

   Check that each component reports the expected messages.

   To get more detailed messages, you can adjust the logging level of some components. For example, you can set the logging levels of OMEGAMON Data Broker and OMEGAMON Data Connect.
2. Check for common issues.

   Tip:

   - If possible, before introducing SSL/TLS (security protocols), test that your configuration works without SSL/TLS. For example, in a sandbox environment that is entirely inside a secure intranet.
   - Check that you are using the correct character encoding for each configuration member. For details, see “Configuration” on page 89.
   - As a rudimentary test that OMEGAMON Data Connect is receiving the expected data from OMEGAMON Data Broker, temporarily enable the STDOUT output of OMEGAMON Data Connect.
3. Finally, if you cannot resolve the problem, gather diagnostic information, and then contact IBM Software Support.

Related reference

Messages
Each component involved in OMEGAMON Data Provider writes messages that describe activity or errors.

Expected messages
These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Related information

Data not arriving at a destination analytics platform

Common issues

Before contacting IBM Software Support, check for these common issues.

Data not arriving at a destination analytics platform

Symptoms

An analytics platform doesn’t show data from OMEGAMON Data Provider. For example, you're expecting an analytics platform to show charts populated with data from OMEGAMON Data Provider, but the charts are empty, with no data.

Causes

There are numerous possible causes across multiple components.

Resolving the problem

1. Eliminate configuration issues with the destination analytics platform as the cause.

Troubleshooting configuration issues with destination analytics platforms is beyond the scope of this OMEGAMON Data Provider documentation.

Tip: Use a generic "network listener" application, separate from the analytics platform, to check whether data is arriving at the destination *computer* in the expected format at the expected port.

2. Follow the general troubleshooting steps for OMEGAMON Data Provider.

Related tasks

Troubleshooting

Follow these steps to diagnose and correct problems that you experience with OMEGAMON Data Provider. For example, expected data not arriving at a destination analytics platform.

OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

Symptoms

The address space where an OMEGAMON collection task is running (typically, the monitoring agent address space) is missing one or both of the following expected messages for a table (attribute group):

KPQH037I
Reports that the collection task has written attributes to the OMEGAMON persistent data store (PDS).

KPQH038I
Reports that the collection task has sent attributes to OMEGAMON Data Broker.

Causes

- The historical data collection for this table might not be correctly configured in OMEGAMON.
- The table might not be correctly specified in the OMEGAMON Data Provider collection configuration member, RKANPARU(KAYOPEN).
- The monitoring agent might require additional configuration to collect this table.

Resolving the problem

1. Check that the historical data collection for this table has been created and activated (distributed).
2. Check that there is a corresponding entry in RKANPARU(KAYOPEN) for this table. Check that the entry selects the collection interval specified for the collection.
3. Check whether the monitoring agent requires additional configuration to collect this table. For details, see the monitoring agent documentation.

   Some examples (not comprehensive):

   IBM Z OMEGAMON for CICS
   To collect bottleneck analysis data, you need to start internal bottleneck collection. Either set the configuration parameter **BOTTLENECK_ANALYSIS** to AUTO or use a command or user interface to manually activate collection.

   IBM OMEGAMON for Messaging on z/OS
   For some tables, you need to set the configuration parameter **KMQ_HISTCOLL_DATA_FLAG** to YES.

   IBM Z OMEGAMON AI for Networks
   To collect z/OS Encryption Readiness Technology (zERT) data, you need to set the configuration parameter **KN3_TCP_ZERT** to Y.

4. If the issue is not resolved, contact IBM Software Support.

Related tasks

Starting OMEGAMON Data Provider

Starting OMEGAMON Data Provider involves starting the related components: OMEGAMON Data Connect, OMEGAMON Data Broker, and the data source, such as the OMEGAMON runtime environment.

Adding more OMEGAMON collections to OMEGAMON Data Provider

If you have already configured an OMEGAMON runtime environment to send collections to OMEGAMON Data Provider, then follow the steps here to add more.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

Expected messages
These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Related information

KPQH037I
TABLE *table* HAS BEEN CONNECTED TO PDS

KPQH038I
TABLE *table* HAS BEEN CONNECTED TO BROKER

OMEGAMON Data Connect fails with `charset.MalformedInputException`

Symptoms

The OMEGAMON Data Connect log contains the following message:

```
hh:mm:ss.SSS [main] ERROR org.springframework.boot.SpringApplication -
Application run failed
org.yaml.snakeyaml.error.YAMLException:
java.nio.charset.MalformedInputException: Input length = 1
```

Causes

The OMEGAMON Data Connect configuration file `config/connect.yaml` is incorrectly encoded. The file must be encoded in UTF-8.

Example incorrect encodings:

- EBCDIC
- ISO8559-1, where the file includes byte values that are valid in ISO8559-1 but invalid in UTF-8

Resolving the problem

Ensure that the file is valid UTF-8.

For compatibility with common z/OS UNIX tools and applications, the sample `connect.yaml` is supplied on z/OS UNIX tagged as being encoded in ISO8559-1 (CCSID 819).

The supplied sample file only uses ASCII characters. ASCII characters have 7-bit byte values; byte values under 128. In this case, there is no difference between ISO8859-1 and UTF-8, because both encodings are supersets of ASCII. However, outside of the common subset of ASCII characters, byte values that are valid in ISO8859-1 can be invalid in UTF-8.

If you use an editor that interprets and writes the file using ISO8859-1, **only use ASCII characters**. Otherwise, you could insert byte values that are invalid in UTF-8.

For example, in ISO8859-1, the byte value X'A9' represents the copyright symbol (©). However, in UTF-8, X'A9' is valid only as a continuation byte in a multi-byte sequence. If you insert a copyright symbol in an editor that uses ISO8859-1, then the file will be invalid UTF-8. Instead, to insert a copyright symbol, your editor must use UTF-8, which will insert the correct 2-byte sequence X'C2A9'.

Related reference

OMEGAMON Data Connect configuration parameters
OMEGAMON Data Connect configuration parameters identify inputs, such as the TCP port on which to listen for data from OMEGAMON Data Broker, and outputs, such as destination analytics platforms. You can filter which data to output.

OMEGAMON Data Connect fails with UnsupportedClassVersionError

Symptoms

The OMEGAMON Data Connect log contains the following message:

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/
springframework/boot/loader/PropertiesLauncher has been compiled by a more
recent version of the Java Runtime (class file version x1.y1), this version
of the Java Runtime only recognizes class file versions up to x2.y2
```

Causes

You tried to run OMEGAMON Data Connect using an old version of Java.

Resolving the problem

Upgrade the Java runtime environment that you use to run OMEGAMON Data Connect to Java 17 or later, 64-bit edition.

If you are using the Java Batch Launcher and Toolkit for z/OS (JZOS) to run OMEGAMON Data Connect, then check that your JCL refers to the correct version-specific Java virtual machine (JVM) load module name, JVMLDM*xx*.

Check that the JAVA_HOME environment variable refers to the correct version of Java .

Related tasks

Upgrading OMEGAMON Data Connect
Upgrade the Java runtime environment that you use to run OMEGAMON Data Connect to Java 17 or later, 64-bit edition. Consider increasing the heap size of the Java virtual machine that you use to run OMEGAMON Data Connect. If you use OMEGAMON Data Connect to send data to Apache Kafka, revise the Kafka output configuration parameters.

OMEGAMON Data Connect fails due to unknown configuration parameters

Symptoms

The OMEGAMON Data Connect log contains the following message reporting an unknown configuration parameter ("Property ... unbound"):

```
APPLICATION FAILED TO START
…
Property: <dotted_key>
…
Reason: The elements [<dotted_key>,<dotted_key2>,…] were left unbound
```

Causes

The OMEGAMON Data Connect configuration file contains one or more parameters that are unknown in the context in which they are specified.

The parameters might be misspelled or incorrectly indented.

An example of incorrect indenting:

```
connect:
  output:
```

```
    stdout:
    enabled: true # Incorrectly indented
```

It's likely that these parameters were intended to enable output to STDOUT. Instead, they cause the problem described here, because the `enabled` key is unknown at the same indent level as `stdout`. In the context of these parameters, `enabled` should be indented as a child of `stdout`.

Resolving the problem

1. Edit the configuration file to fix the problem.
2. Use the OMEGAMON Data Provider configuration validator to validate the updated configuration file.
3. Start OMEGAMON Data Connect.

Related reference

Configuration validator

The OMEGAMON Data Provider configuration validator is a command-line tool that you can use to validate the OMEGAMON Data Provider configuration files that are in YAML format: the OMEGAMON Data Connect configuration file and, if you are using OMEGAMON monitoring agents as a data source, the OMEGAMON Data Provider collection configuration member, `RKANPARU(KAYOPEN)`.

Gathering diagnostic information

Before you report a problem with OMEGAMON Data Provider to IBM Software Support, you need to gather the appropriate diagnostic information.

About this task

The following procedure lists the information that you need to gather and then send to IBM Software Support to help diagnose a problem.

Procedure

1. Write a clear description of the problem and the steps to reproduce the problem.
2. Gather the configuration parameters for each component of OMEGAMON Data Provider.

   `RKANPARU(KAYOPEN)`
   If you are using OMEGAMON monitoring agents as a data source: the OMEGAMON collection configuration

   `PARMLIB(KAYSIP`*`xx`*`)`
   Zowe cross-memory server configuration, containing OMEGAMON Data Broker configuration parameters

   `config/connect.yaml`
   OMEGAMON Data Connect configuration in the OMEGAMON Data Connect user directory
3. Gather the complete job log and any dumps from each of the z/OS address spaces involved.
   - If you are using OMEGAMON monitoring agents as a data source: the address spaces where the OMEGAMON collection tasks are running. For example, for the z/OS monitoring agent: the z/OS monitoring server address space.
   - The Zowe cross-memory server that is running OMEGAMON Data Broker.
   - OMEGAMON Data Connect, if you are running it on z/OS.

   Store each job log and dump in a separate text file with a semantic (meaningful, plain English) name that identifies its contents (for example, include in the file names the terms "collection", "broker", "connect").

   Tip: In z/OS SDSF, to save the complete job log to a data set, enter the action XD next to the job.

4. If you are running OMEGAMON Data Connect on a distributed platform (off z/OS), gather the Java log, including the `stdout` and `stderr` file contents.
5. Specify the operating systems and versions involved.
   - z/OS version
   - If you are running OMEGAMON Data Connect off z/OS, the corresponding details for that platform, such as the operating system distribution name and version
6. Specify the Java version that you are using to run OMEGAMON Data Connect.

   Tip: To get the Java version, use the command `java -version`.
7. Specify details of the analytics platform or application to which you are sending data.

   Examples:
   - The name and version of the analytics platform.
   - The operating system distribution name and version.
   - How you have configured the analytics platform to ingest data from OMEGAMON Data Connect. For example, for the Elastic Stack: the Logstash configuration and index template; for Splunk, the configuration stanzas.
   - Whether, and how, you have tested that the destination is correctly configured to ingest data, independent from OMEGAMON Data Provider. For example, have you used a stand-alone TCP forwarder to send a sample line of JSON to the destination, in the same format sent by OMEGAMON Data Connect?

Messages

Each component involved in OMEGAMON Data Provider writes messages that describe activity or errors.

Message location and prefix by component

Component	Message location	Message prefix
Data source: OMEGAMON monitoring agents Other data sources: see the separate documentation for that software	RKLVLOG output data set of the corresponding job for the agent. For example, for attributes from IBM Z OMEGAMON AI for z/OS: the monitoring server job (default job name: OMEGDS).	KAYL, KPQD, KPQH
OMEGAMON Data Broker	SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker or, for some messages, the z/OS system log. The Zowe cross-memory server also writes its own messages, with the prefix ZWES. For details, see the message descriptions in the Zowe documentation.	KAYB
OMEGAMON Data Connect	STDOUT file. If you are running OMEGAMON Data Connect as a z/OS job: the STDOUT output data set of that job.	KAYC

Message format

Each OMEGAMON Data Provider message begins with an identifier in the following format:

KAY*xnnnns*

or

KPQ*xnnns*

where:

KAY*x*
Identifies the origin of the message as one of the following components:

KAYL
OMEGAMON historical collection task. See also KPQ*x*.

KAYB
OMEGAMON Data Broker.

KAYC
OMEGAMON Data Connect.

KPQ*x*
Identifies the origin of the message as a historical collection task (*x*: D or H).

nnnn* or *nnn
4-digit or 3-digit message identification number.

s
Severity of the message:

I
Informational.

W
Warning to alert you to a possible error condition.

E
Error.

The documentation for each message includes the following information:

Explanation
Describes what the message text means, why the message occurred, and what its variables represent.

System action
Describes what the system will do in response to the event that triggered this message.

User response
Describes whether a response is necessary, what the appropriate response is, and how the response will affect the system or program.

Related tasks

Troubleshooting
Follow these steps to diagnose and correct problems that you experience with OMEGAMON Data Provider. For example, expected data not arriving at a destination analytics platform.

Expected messages

These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If data is not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

A missing expected message indicates a problem. However, the cause of the problem is not necessarily at the point in processing where the message should occur. The cause might be *upstream*.

Components and their messages are presented here according to the direction of data flow: from the data source, to OMEGAMON Data Broker, and then to OMEGAMON Data Connect. The actual chronological order of some messages can differ from the order presented here.

Tip: Solving a problem upstream can solve multiple problems downstream. Investigate missing messages in the order presented here.

Messages from OMEGAMON Data Provider might be interleaved with messages from other software, such as the operating system, a related component, or a supporting software framework. For example:

- The STDOUT file for OMEGAMON Data Connect includes messages from the Spring framework.
- The SYSPRINT output data set of the Zowe cross-memory server includes ZWE-prefix messages from the server that are not specific to the OMEGAMON Data Broker plug-in.

Data source

If you are using OMEGAMON monitoring agents as a data source, then the RKLVLOG output data set of each agent job (for example, job names OM*) should contain the following messages.

Message	Description
KAYL0005I KPQHST*xx*: BROKER NAME = '*value*'	Echoes the *value* of the broker.name key in the RKANPARU(KAYOPEN) configuration member. **If this message is missing**, check that your OMNIMON Base component meets the required APAR level for OMEGAMON Data Provider. For details, see "Prerequisites for OMEGAMON Data Provider" on page 44.

Message	Description
KAYL0005I KPQHST*xx*: PCODE='*product*', TABLE='*table_name*', INTERVAL=*interval*, DEST={*destinations*} ...	Echoes each entry under the `collections` key in RKANPARU(KAYOPEN). **If this message is missing** for a table, check that there is a corresponding entry for the table under the `collections` key.
KPQH037I KPQHSMGR: TABLE *product.table_name* HAS BEEN CONNECTED TO PDS ...	Reports the first instance of a record for each table written to the PDS. This message is written only for tables that are explicitly specified under the `collections` key. If RKANPARU(KAYOPEN) does not explicitly specify a table, then the default behavior is to write records from the table to PDS without reporting this message. **If this message is missing** for a table, see "OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS" on page 178.
KPQH038I KPQHSMGR: TABLE *product.table_name* HAS BEEN CONNECTED TO BROKER ...	Reports the first instance of a record for each table sent to OMEGAMON Data Broker. **If this message is missing** for a table, see "OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS" on page 178.

For information about messages from other data sources, see the separate documentation for that software.

OMEGAMON Data Broker

The SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker (for example, job name KAYSIS01) should contain the following messages.

Message	Description
KAYB0005I CIDB starting, *version* (APAR *apar_number*, *build_time_stamp*)	Reports that OMEGAMON Data Broker is starting. Also reports the OMEGAMON Data Broker version and APAR number. Some messages use the term CIDB. CIDB is an abbreviation of Common Intercept Data Broker. CIDB is a synonym for OMEGAMON Data Broker.
KAYB0009I Init step 'CIDB anchor initialization' done	Normal initialization message.
KAYB0016I No CIDB ID has been provided	Normal initialization message. OMEGAMON Data Provider users do not need to provide this ID.
KAYB0016I Forwarder subsystem component is on	Corresponds to the OMEGAMON Data Broker configuration parameter KAY.CIDB.FWD=ON.

Message	Description
KAYB0009I Init step 'Load CIDB parameters' done KAYB0009I Init step 'CIDB global area initialization' done KAYB0009I Init step 'CIDB ID generation' done, ID = '*cidb_id*' KAYB0009I Init step 'CIDB store manager creation' done KAYB0020I Store '*store_name*' has been added KAYB0009I Init step 'User defined store creation' done KAYB0009I Init step 'Forwarder subsystem initialization' done	Normal initialization messages.
KAYB0036I Forwarder *forwarder_id* has connected to sink *host*:*port*	OMEGAMON Data Broker has connected to OMEGAMON Data Connect.
KAYB0009I Init step 'Forwarder subsystem startup' done KAYB0006I CIDB successfully started	Normal initialization messages.

OMEGAMON Data Connect

The STDOUT file of OMEGAMON Data Connect should contain the following messages.

General messages, regardless of which outputs are enabled:

Message	Description
KAYC0026I Creating JSON mapping provider	Normal initialization message
KAYC0023I Starting TCP input service listening on *hostname*:*port*	OMEGAMON Data Connect has started listening on *hostname*:*port* for TCP input from OMEGAMON Data Broker.
KAYC0028I Source *hostname*:*port* has connected	The instance of OMEGAMON Data Broker that is at *hostname*:*port* has connected to OMEGAMON Data Connect.
KAYC0038I Starting console listener	OMEGAMON Data Connect has started listening for console commands. For example, z/OS MVS system **MODIFY** commands.
KAYC0035I Build: *build_identifier*	Reports the OMEGAMON Data Connect build identifier.
KAYC0067I Framework version = *connect_framework_version*	Identifies the version of the OMEGAMON Data Connect framework that was used to build the OMEGAMON Data Connect core JAR file, odp-server-*version*.jar.
KAYC0068I Extension file *jar_file_path* found (spec=*framework_version*, impl=*mapping_extension_version*)	OMEGAMON Data Connect reports this message for each mapping extension JAR file that it finds in the directories that are listed in the OMEGAMON Data Connect runtime option -Dodp.ext.

Message	Description
KAYC0069I *class_count* mapping classes found in *jar_file_path*	OMEGAMON Data Connect reports the number of mapping classes it finds in each mapping extension JAR file.
KAYC0008I Creating mapping class for table *table_name*	Indicates the first instance of a record received for this table.
KAYC0033I Table *table_name* received from *origin_type origin_name*	Indicates the first instance of a record received for this table *from this origin*.
...	The origin type depends on the table. Examples: sysplex, CICS region.
KAYC0036I *filter_scope* filter selected table: *table_name*, fields: *field_list*	OMEGAMON Data Connect has been configured to filter records of this table.
...	

If STDOUT output is enabled:

Message	Description
KAYC0024I Starting STDOUT output service	Normal initialization message.

If TCP output is enabled:

Message	Description
KAYC0009I Starting TCP output service	Normal initialization message.
KAYC0042I Starting TCP sink: *sink_name* {host: *hostname*, port: *port*} KAYC0010I Connecting to TCP sink: *sink_name* {host: *hostname*, port: *port*} KAYC0011I Connected to TCP sink: *sink_name* {host: *hostname*, port: *port*} ...	Normal initialization messages for each sink.

If HTTP output is enabled:

Message	Description
KAYC0076I Starting HTTP output service	Normal initialization message.
KAYC0078I Starting output to HTTP endpoint: *endpoint_name* {url: *url*}	Normal initialization message for each endpoint.
...	

If Prometheus output is enabled:

Message	Description
KAYC0018I Starting metrics service	Normal initialization message.
KAYC0037I Registered metric for table: *table_name*, field: *field_name*, type: *metric_type*, labels: [*label_list*]	Normal initialization message for each metric.

Message	Description
...	

If Kafka output is enabled:

Message	Description
KAYC0025I Starting Kafka output service	Normal initialization message.

When OMEGAMON Data Connect stops:

Message	Description
KAYC0034I Stopping server KAYC0027I Stopping TCP listener KAYC0029I Source *hostname*:*port* has disconnected	Normal shutdown messages.
KAYC0032I Stopping TCP output service	If TCP output was enabled.
KAYC0043I Stopping TCP sink: *sink_name* {host: *hostname*, port: *port*}	For each TCP output sink.
KAYC0077I Stopping HTTP output service	If HTTP output was enabled.
KAYC0081I Stopping output to HTTP endpoint: *endpoint_name* {url: *url*}	For each HTTP output endpoint.

Related tasks

Troubleshooting
Follow these steps to diagnose and correct problems that you experience with OMEGAMON Data Provider. For example, expected data not arriving at a destination analytics platform.

Related information

OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

KAYL, KPQD, KPQH: Messages from OMEGAMON collection tasks

Messages with the prefix KAYL, KPQD, or KPQH are from the OMEGAMON historical collection tasks that send attributes to OMEGAMON Data Broker.

The KPQD and KPQH messages documented here are the messages introduced by OMEGAMON Data Provider. For descriptions of other KPQ-prefix messages, see the OMEGAMON shared documentation.

KAYL0001E ***task*: *resource* NOT ALLOCATED**

Explanation

The OMEGAMON historical collection task could not allocate the *resource* due to memory shortage.

The *task* is the name of the task in which the error originated, in the format KPQHST*pp*, where *pp* is the product code.

System action:
Processing of historical data and streaming stops for the application identified by task.

User response:
Contact IBM Software Support. See the diagnostic information in the ITMS:Engine log, RKLVLOG.

KAYL0002W ***task*: MEMBER *member_name* READ ERROR, RC = *rc*, RSN = *rsn***

Explanation

The OMEGAMON historical collection task could not read the configuration member *member_name*.

The *task* is the name of the task in which the error originated, in the format KPQHST*pp*, where *pp* is the product code.

The return code *rc* and reason code *rsn* indicate the cause of the error. These codes are from the z/OS MVS

assembler logical parmlib support service, IEFPRMLB. For descriptions of IEFPRMLB return codes and reason codes, see z/OS documentation.

System action:
Processing of historical data and streaming stops for the application identified by *task* until the issue has been resolved.

User response

1. Fix the issues reported in the message.
2. Reload the configuration by entering the following MVS system command:

   ```
   MODIFY jobname,KPQ,RELOAD_CONFIG,KAY
   ```

3. If you cannot resolve the issue, contact IBM Software Support.

KAYL0003W ***task*: MEMBER *member_name* NOT FOUND, OPEN DATA PROCESSING IS STOPPED**

Explanation

The configuration member *member_name* is missing.

The *task* is the name of the task in which the error originated, in the format KPQHST*pp*, where *pp* is the product code.

The *rc* and *rsn* are from the IEFPRMLB service and indicate the cause of the error.

System action:
Streaming stops for the application identified by *task* until the issue has been resolved.

User response

1. Deploy the missing configuration member.
2. Load the configuration by entering the following MVS system command:

   ```
   MODIFY jobname,KPQ,RELOAD_CONFIG,KAY
   ```

KAYL0004W ***task*: YAML CONFIG ERROR, *details***

Explanation

An error occurred while processing the YAML configuration member.

The *task* is the name of the task in which the error originated, in the format KPQHST*pp*, where *pp* is the product code.

The *details* contain additional information such as the error type and line number.

The following details:

```
PARSER FAILED WITH CODE 2 AT LINE 0, COLUMN 0
(invalid trailing UTF-8 octet)
```

indicate that the character encoding of the member might be incorrect. For example, this error occurs if the member contains square brackets ([ ]) encoded using EBCDIC code page 037. The member must be encoded using EBCDIC code page 1047.

System action:
Depending on the error details, either the entire configuration or parts of the configuration are ignored.

User response

1. Correct the error by editing the YAML configuration member according to the provided details.
2. Reload the configuration by entering the following MVS system command:

   ```
   MODIFY jobname,KPQ,RELOAD_CONFIG,KAY
   ```

KAYL0005I ***task*: *parameters***

Explanation

Information about collection configuration parameters set by the YAML configuration member.

The *task* is the name of the collection task to which the parameter applies.

The *parameters* report parameters set by the YAML configuration member.

System action:
The configuration parameters are applied to the collection task.

User response:
None required.

Related reference

Configuration parameters for OMEGAMON monitoring agents as a data source
OMEGAMON runtime environment member RKANPARU(KAYOPEN) configures the collection tasks of OMEGAMON monitoring agents. The member contains configuration parameters that select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

KPQD107E **KPQDBCMD: KPQ VECTOR NOT FOUND**

Explanation:
While running the **KPQ** operator command, the KPQ vector was not found.

System action:
The **KPQ** command terminates.

User response:
View the related messages in the ITMS:Engine log, RKLVLOG. Contact IBM Software Support.

KPQD108E **KPQDBCMD: MODULE KPQSPCMD NOT AVAILABLE, RC = *rc***

Explanation:
While running the **KPQ** operator command, the command handler module was not available.

System action:
The **KPQ** command terminates.

User response:
View the related messages in the ITMS:Engine log, RKLVLOG. Contact IBM Software Support.

KPQH032W **KPQHSMGR: BROKER MODULE *name* NOT LOADED, RC = *rc*, RSN = *rsn***

Explanation:
The broker API module *name* could not be loaded. The return code (*rc*) and reason (*rsn*) values have the abend and reason codes from the **LOAD** system call.

System action:
No data is sent to the broker.

User response:
Review the JCL for the job that runs the OMEGAMON historical collection task. Check that the broker module is in the STEPLIB data sets specified by the JCL. If you cannot resolve the issue, contact IBM Software Support.

KPQH033W **KPQSPCMD: COMMAND IGNORED, *reason***

Explanation:
A **MODIFY** command has been entered for the job that runs the OMEGAMON historical collection task; for example, the monitoring server job. The **MODIFY** command has been ignored. The *reason* specifies the cause of the error.

System action:
The command is ignored.

User response:
Enter a correct **MODIFY** command.

KPQH034I **KPQSPCMD: COMMAND ACCEPTED, *details***

Explanation:
A **MODIFY** command has been entered for the job that runs the OMEGAMON historical collection task; for example, the monitoring server job. The **MODIFY** command has been accepted. The *details* contain additional command response information.

System action:
The command is accepted.

User response:
None required.

KPQH037I **TABLE *table* HAS BEEN CONNECTED TO PDS**

Explanation
The OMEGAMON historical collection task has successfully written a record of this table to the persistent data store (PDS).

This message is written only if the table is explicitly specified in the RKANPARU(KAYOPEN) configuration member. If the member does not exist, or the member exists but does not explicitly specify the table, then the default behavior is to write records from the table to PDS without reporting this message.

This message is written only in the following situations:

- For the first instance of a record of this table since the task's configuration was loaded: either when the task's job started or when the configuration was reloaded by a **MODIFY** command while the job was running.
- After the issue that caused message KPQH039W has been fixed.

Tip: The frequency of incoming data is determined by the collection interval of the collection for this table. A long collection interval can mean a long delay before this message occurs.

System action:
None.

User response:
None required.

Related information
OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

KPQH038I **TABLE *table* HAS BEEN CONNECTED TO BROKER**

Explanation
The OMEGAMON historical collection task has successfully sent a record from this table to OMEGAMON Data Broker.

This message is written only if the table is explicitly specified in the RKANPARU(KAYOPEN) configuration member. If the member does not exist, or the member exists but does not explicitly specify the table, then the default behavior is to write records from the table to PDS without reporting this message.

This message is written only in the following situations:

- For the first instance of a record of this table since the task's configuration was loaded: either when the task's job started or when the configuration was reloaded by a **MODIFY** command while the job was running.
- After the issue that caused message KPQH040W has been fixed.

Tip: The frequency of incoming data is determined by the collection interval of the collection for this table. A long collection interval can mean a long delay before this message occurs.

System action:
None.

User response:
None required.

Related information

OMEGAMON attributes not arriving at OMEGAMON Data Broker or PDS

KPQH039W **PDS CONNECTION FOR TABLE *table* FAILED, *reason***

Explanation

The OMEGAMON historical collection task failed to write records of this table to the persistent data store (PDS). The *reason* provides details of the cause.

This message is written only if the table is explicitly specified in the RKANPARU(KAYOPEN) configuration member. If the member does not exist, or the member exists but does not explicitly specify the table, then the default behavior is to write records from the table to PDS without reporting this message.

System action:
Until this issue is resolved, no records of this table are written to the PDS.

User response:
Review the provided details and take appropriate action. If you cannot resolve the issue, contact IBM Software Support.

KPQH040W **BROKER CONNECTION FOR TABLE *table* FAILED, *reason***

Explanation

The OMEGAMON historical collection task failed to send records of this table to OMEGAMON Data Broker. The *reason* provides details of the cause.

This message is written only if the table is explicitly specified in the RKANPARU(KAYOPEN) configuration member. If the member does not exist, or the member exists but does not explicitly specify the table, then the default behavior is to write records from the table to PDS without reporting this message.

System action:
Until this issue is resolved, no records of this table are sent to OMEGAMON Data Broker.

User response

Review the provided details and take appropriate action.

reason values and suggested actions:

STORE NOT FOUND
Ensure that the OMEGAMON store is defined in the OMEGAMON Data Broker configuration member.

BROKER HAS NO CONNECTION TO SINK
Ensure that OMEGAMON Data Broker is connected to OMEGAMON Data Connect.

BROKER OFFLINE
Ensure that the OMEGAMON Data Broker name is correct in the collection configuration member.

Ensure that the Zowe cross-memory server that hosts OMEGAMON Data Broker is running.

RC = *rc*, RSN = *rsn*
Contact IBM Software Support.

If you cannot resolve the issue, contact IBM Software Support.

KPQH041E ***task*: CONFIG NOT LOADED, HISTORY/OPEN DATA PROCESSING IS STOPPED**

Explanation

There are issues with the OMEGAMON historical collection task configuration member, *rte_hilev*.*rte_name*.RKANPARU(KAYOPEN), for the application identified by *task*.

The *task* is the name of the task in which the error originated, in the format KPQHST*pp*, where *pp* is the product code.

System action:
Until the issue is resolved, processing of records from the application (*pp*) stops. No records of tables from this application are sent to OMEGAMON Data Broker or written to PDS.

User response:
Review the issues reported in previous error messages. If you cannot resolve these issues, contact IBM Software Support.

KPQH042W ***task*: CONFIG NOT LOADED, EXISTING CONFIG WILL BE USED**

Explanation

A **MODIFY** command has been entered for the job that runs the OMEGAMON historical collection

task, to reload the configuration. However, the new configuration is ignored. Processing of historical data and/or streaming continues the same as before the **RELOAD_CONFIG** command was issued; the new parameters are ignored.

The *task* is the name of the task in which the error originated, in the format KPQHST*pp*, where *pp* is the product code.

System action:
The new configuration is ignored. Processing of historical data and streaming continues the same as before for the application identified by *task* until the issue has been resolved.

User response:
Address the issues reported in previous error messages. If you cannot resolve this issue, contact IBM Software Support.

KAYB: Messages from OMEGAMON Data Broker

Messages with the prefix KAYB are from OMEGAMON Data Broker.

Messages with the prefix KAYBN are from network functions of OMEGAMON Data Broker, such as secure connection (SSL/TLS) functions.

OMEGAMON Data Broker writes messages to the SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker or, for some messages, the z/OS system log.

The Zowe cross-memory server also writes its own messages, with the prefix ZWES. For details, see the message descriptions in the Zowe documentation.

Some messages use the term CIDB. CIDB is an abbreviation of Common Intercept Data Broker. CIDB is a synonym for OMEGAMON Data Broker.

KAYB0001I ***trace_message***

Explanation:
OMEGAMON Data Broker trace message.

System action:
None.

User response:
None required.

KAYB0002I ***trace_message***

Explanation:
OMEGAMON Data Broker service trace message.

System action:
None.

User response:
None required.

KAYB0003I ***trace_dump***

Explanation:
OMEGAMON Data Broker trace dump.

System action:
None.

User response:
None required.

KAYB0004I ***command_response***

Explanation:
Response from an operator command to OMEGAMON Data Broker.

System action:
None.

User response:
None required.

KAYB0005I **CIDB starting, version *version* (APAR *apar_number*, *build_time_stamp*)**

Explanation
OMEGAMON Data Broker initialization has begun.

The message details include the OMEGAMON Data Broker version and APAR number.

System action:
OMEGAMON Data Broker initialization continues.

User response:
None required.

KAYB0006I **CIDB successfully started**

Explanation:
OMEGAMON Data Broker has successfully initialized.

System action:
OMEGAMON Data Broker is ready to accept service calls.

User response:
None required.

KAYB0007I **CIDB terminating**

Explanation:
OMEGAMON Data Broker termination has begun.

System action:
OMEGAMON Data Broker termination continues.

User response:
None required.

KAYB0008I CIDB successfully terminated

Explanation:
OMEGAMON Data Broker has successfully terminated.

System action:
None. OMEGAMON Data Broker has stopped.

User response:
None required.

KAYB0009I Init step '*description*' done

Explanation:
This OMEGAMON Data Broker initialization step has successfully completed.

System action:
OMEGAMON Data Broker initialization continues.

User response:
None required.

KAYB0010W Init step '*description*' failed -

Explanation:
A failure occurred during this OMEGAMON Data Broker initialization step.

System action:
Depending on the step, some functionality might be disabled. OMEGAMON Data Broker initialization continues.

User response:
If you cannot resolve the issue, contact IBM Software Support.

KAYB0011E Init step '*description*' failed -

Explanation:
A severe failure occurred during this OMEGAMON Data Broker initialization step.

System action:
OMEGAMON Data Broker initialization stops.

User response:
If you cannot resolve the issue, contact IBM Software Support.

KAYB0012I Term step '*description*' done

Explanation:
This OMEGAMON Data Broker termination step successfully completed.

System action:
OMEGAMON Data Broker termination continues.

User response:
None required.

KAYB0013W Term step '*description*' failed -

Explanation:
A failure occurred during this OMEGAMON Data Broker termination step.

System action:
Depending on the step, some functionality might not be terminated cleanly. OMEGAMON Data Broker termination continues.

User response:
If you cannot resolve the issue, contact IBM Software Support.

KAYB0014E Term step '*description*' failed -

Explanation:
A severe failure occurred during this OMEGAMON Data Broker termination step.

System action:
OMEGAMON Data Broker termination stops. Some components might not be terminated properly.

User response:
If you cannot resolve the issue, contact IBM Software Support.

KAYB0015W CIDB modify command error -

Explanation:
An error occurred handling a **MODIFY** command for OMEGAMON Data Broker.

System action:
OMEGAMON Data Broker does not perform the action requested by the **MODIFY** command.

User response:
Review the details provided and then retry the command.

KAYB0016I *response*

Explanation:
This message describes the effect of, or response to, an OMEGAMON Data Broker configuration parameter.

System action:
OMEGAMON Data Broker continues normal processing.

User response:
None required.

KAYB0017W CIDB parameter error -

Explanation:
An error occurred handling an OMEGAMON Data Broker configuration parameter.

System action:
OMEGAMON Data Broker ignores the parameter.

User response:
Review the details and correct the parameter.

KAYB0018E CIDB ID not generated -

Explanation:
An error occurred generating the OMEGAMON Data Broker ID.

System action:
OMEGAMON Data Broker initialization stops. The OMEGAMON Data Broker service will not be available.

User response:
Use the **KAY.CIDB.ID** configuration parameter to specify an OMEGAMON Data Broker ID, rather than relying on an automatically generated value.

KAYB0019W Store configuration error -

Explanation:
An error occurred configuring the OMEGAMON Data Broker store.

System action:
OMEGAMON Data Broker ignores the affected store parameter.

User response:
Review the details and ensure that the parameters are correct.

KAYB0020I Store '*store_name*' has been added

Explanation:
The OMEGAMON Data Broker store has successfully initialized.

System action:
OMEGAMON Data Broker initialization continues. The store will be available when OMEGAMON Data Broker initialization is complete.

User response:
None required.

KAYB0021W Store '*store_name*' has not been added, RC = *return_code*

Explanation:
An error occurred initializing this OMEGAMON Data Broker store.

System action:
OMEGAMON Data Broker initialization continues. However, this store will not be available.

User response:
return_code 35 indicates a duplicate store name: correct the store name in the PARMLIB(ZWESIP*xx*) configuration member. For other *return_code* values, contact IBM Software Support.

KAYB0022E *subsystem_name* subsystem error -

Explanation
An error occurred initializing this OMEGAMON Data Broker subsystem.

If the error is specific to a particular forwarder, then the details following the fixed message text include the forwarder ID.

System action
OMEGAMON Data Broker initialization continues. However, the subsystem, or part of the subsystem, will be unavailable. For example, if the message refers to the forwarder subsystem:

- If the details following the fixed message text cite a forwarder ID, then only that forwarder will be unavailable. Other forwarders are unaffected.
- If the details following the fixed message text do not cite a forwarder ID, then the entire forwarder subsystem will be unavailable. No forwarders will be available.

User response:
If you cannot resolve the issue, contact IBM Software Support.

KAYB0023W *subsystem_name* subsystem configuration error -

Explanation:
An error occurred configuring this OMEGAMON Data Broker subsystem.

System action:
OMEGAMON Data Broker ignores the affected parameter.

User response:
Review the details and ensure that the parameters are correct.

KAYB0036I Forwarder *forwarder_id* has connected to sink *host*:*port*

Explanation:
OMEGAMON Data Broker has successfully connected to a sink, such as OMEGAMON Data Connect.

System action:
OMEGAMON Data Broker sends data to the sink.

User response:
None required.

KAYB0037I Forwarder *forwarder_id* has disconnected from sink *host*:*port*

Explanation:
A sink, such as OMEGAMON Data Connect, has disconnected from OMEGAMON Data Broker.

System action:
OMEGAMON Data Broker frees resources that were allocated to that sink.

User response:
None required.

KAYB0038W **Forwarder *forwarder_id* has failed to connect to sink *host*:*port***

Explanation:
OMEGAMON Data Broker has failed to connect to a sink, such as OMEGAMON Data Connect.

System action:
OMEGAMON Data Broker retries connection. Failed retries are not reported. Message KAYB0036I indicates a successful retry.

User response:
Review SYSPRINT for other warning or error messages that might be related to this warning.

KAYB0039E **Config member *member_name* error -**

Explanation:
OMEGAMON Data Broker encountered an error reading the configuration member.

System action:
OMEGAMON Data Broker ignores all of the parameters in the configuration member.

User response:
If you cannot resolve the issue, contact IBM Software Support.

KAYB0040E **CIDB startup failed**

Explanation:
OMEGAMON Data Broker failed to initialize.

System action:
OMEGAMON Data Broker stops.

User response:
Review the preceding messages. If you cannot resolve the issue, contact IBM Software Support.

KAYB0041E **CIDB terminated with errors**

Explanation:
OMEGAMON Data Broker unsuccessfully terminated.

System action:
OMEGAMON Data Broker stops with errors.

User response:
Review the preceding messages. If you cannot resolve the issue, contact IBM Software Support.

KAYB0042I **Forwarder '*forwarder_id*' has connected to sink**

Explanation:
OMEGAMON Data Broker has successfully connected to a sink.

System action:
OMEGAMON Data Broker sends data to the sink.

User response:
None required.

KAYB0043I **Forwarder '*forwarder_id*' has disconnected from sink**

Explanation:
OMEGAMON Data Broker has disconnected from a sink.

System action:
OMEGAMON Data Broker frees the resources allocated to the sink.

User response:
None required.

KAYB0044W **Forwarder '*forwarder_name*' has failed to connect to sink**

Explanation:
OMEGAMON Data Broker has failed to connect to a sink.

System action:
The forwarder retries to connect. Further failed attempts will not be reported until there has been a successful connection.

User response:
Ensure that the sink is reachable. Review SYSPRINT for other warning or error messages that might be related to this warning.

KAYB0045I ***modify_command***

Explanation:
OMEGAMON Data Broker has received a **MODIFY** command. This message echoes the command details.

System action:
The **MODIFY** command is printed to SYSPRINT and SYSLOG.

User response:
None required.

KAYB0046W **Record queue limit has been reached for forwarder '*forwarder_id*'**

Explanation:
A OMEGAMON Data Broker forwarder has reached its record queue limit. This message is reported only once per sink connection. When the forwarder reconnects, this will be re-reported when the limit is reached again.

System action:
OMEGAMON Data Broker discards some old records to make room for new records.

User response:
To avoid losing records, increase the value of the OMEGAMON Data Broker configuration parameter RECORD_QUEUE_LIMIT.

KAYB0047W **Forwarder '*forwarder_id*' has lost *n* records in total**

Explanation:
After reaching the record queue limit, a OMEGAMON Data Broker forwarder has lost *n* records since the last successful connection.

System action

OMEGAMON Data Broker continues discarding old records to make room for new records.

Approximately every 5 minutes, if the total number of lost records has increased since the previous instance of this message, OMEGAMON Data Broker issues a new message with the updated total.

User response:
To avoid losing records, increase the value of the OMEGAMON Data Broker configuration parameter `RECORD_QUEUE_LIMIT`.

KAYB0052E **Timer '*timer_name*' failed, RC = *return_code*, RSN = *reason_code* (*description*)**

Explanation:
OMEGAMON Data Broker encountered an error in a timer.

System action:
The functionality associated with the timer might not be available.

User response:
Contact IBM Software Support.

KAYB0053E **Lost record check not set up, RC = *return_code*, RSN = *reason_code***

Explanation:
OMEGAMON Data Broker was unable to set up the mechanism for checking lost records.

System action:
Lost records will not be checked or reported.

User response:
Contact IBM Software Support.

KAYB0054W **Lost record check not deleted, RC = *return_code*, RSN = *reason_code***

Explanation:
OMEGAMON Data Broker was unable to remove the mechanism for checking lost records.

System action:
None.

User response:
Contact IBM Software Support.

KAYB0055I **Default store '*store_name*' has been added**

Explanation:
OMEGAMON Data Broker successfully initialized a default store.

System action:
OMEGAMON Data Broker continues initializing. The store will be available when OMEGAMON Data Broker initialization is complete.

User response:
None required.

KAYB0056E **Default store '*store_name*' has not been added, RC = *return_code***

Explanation:
An error occurred initializing this OMEGAMON Data Broker default store.

System action:
OMEGAMON Data Broker initialization stops. The OMEGAMON Data Broker service will not be available.

User response:
Contact IBM Software Support.

KAYB0057E **Bad plug-in environment - *details***

Explanation:
OMEGAMON Data Broker encountered a problem with ZISDYNAMIC, the dynamic linkage base plug-in of the Zowe cross-memory server. OMEGAMON Data Broker requires the dynamic linkage base plug-in.

System action:
OMEGAMON Data Broker initialization stops. The OMEGAMON Data Broker service will not be available.

User response

The appropriate response depends on the *details*.

Typical *details*:

bad stub version *dl_stub_version* (must be >= *broker_stub_version*)
Use a more recent version of the dynamic linkage base plug-in.

This version of the dynamic linkage base plug-in is not compatible with OMEGAMON Data Broker.

The "stub version" refers to the definitions of Zowe cross-memory server internal data structures. The stub version that was used to compile this dynamic linkage base plug-in is earlier than the stub version that was used to compile the OMEGAMON Data Broker plug-in.

If you are using the Zowe cross-memory server supplied with OMEGAMON Data Provider, then ensure that you are using the server load module, dynamic linkage base plug-in, and OMEGAMON Data Broker plug-in that were all supplied with the same APAR level of OMEGAMON Data Provider.

Otherwise, if you are running OMEGAMON Data Broker in a Zowe cross-memory server in a separate installation of Zowe, then check that the Zowe version is compatible with OMEGAMON Data Broker. OMEGAMON Data Broker requires a server from Zowe 2.12.0 or any later 2.*x*.*x*.

`dynamic linkage vector is NULL`
Check that you have registered the dynamic linkage base plug-in, ZISDYNAMIC, in the Zowe cross-memory server configuration member. For details, see "OMEGAMON Data Broker configuration parameters" on page 95.

If you cannot resolve the issue, contact IBM Software Support.

KAYB0058E **Unsupported cross-memory server version *server_version*, versions *supported_versions* are supported only**

Explanation

This version of the Zowe cross-memory server is not compatible with OMEGAMON Data Broker.

supported_versions uses interval notation with an opening square bracket and a closing parenthesis to specify a *right-open* interval:

$[a, b)$

where the supported versions are: *a* up to, but not including, *b*.

System action:
OMEGAMON Data Broker initialization stops. The OMEGAMON Data Broker service will not be available.

User response:
Use a supported version of the Zowe cross-memory server.

KAYB0059E **Unsupported dynamic linkage plug-in version *version*, versions *supported_versions* are supported only**

Explanation

OMEGAMON Data Broker requires the dynamic linkage base plug-in, ZISDYNAMIC. However, this version of the plug-in is not compatible with OMEGAMON Data Broker.

supported_versions uses interval notation with an opening square bracket and a closing parenthesis to specify a *right-open* interval:

$[a, b)$

where the supported versions are: *a* up to, but not including, *b*.

System action:
OMEGAMON Data Broker initialization stops. The OMEGAMON Data Broker service will not be available.

User response:
Use a supported version of the dynamic linkage base plug-in.

KAYB0060E **Protocol check failed for forwarder '*forwarder_id*' (*event_description*), RC = *return_code*, RSN = *reason_code***

Explanation

The OMEGAMON Data Broker forwarder protocol version check failed.

The *event_description*, *return_code*, and *reason_code* are for use by IBM Software Support.

System action:
The forwarder stops sending data.

User response:
Contact IBM Software Support.

KAYB0061I **Sink of forwarder '*forwarder_id*' can only support version 1 of the forwarder protocol**

Explanation:
OMEGAMON Data Broker detected a sink that only supports version 1 of the OMEGAMON Data Broker forwarder protocol.

System action:
The forwarder uses protocol version 1.

User response:
Update the sink to support version 2 of the OMEGAMON Data Broker forwarder protocol.

KAYB0062W **Bad protocol check response in forwarder '*forwarder_id*', falling back to version 1**

Explanation:
OMEGAMON Data Broker could not determine which versions of the OMEGAMON Data Broker forwarder protocol are supported by the sink.

System action:
The forwarder falls back to using protocol version 1.

User response:
Review the description of the OMEGAMON Data Broker sink parameter `PROTOCOL_VERSION` and consider whether to set a different value.

KAYB0063E **Internal error in forwarder '*forwarder_id*', site = '*location_in_forwarder*', RC=*return_code***

Explanation
An internal error occurred in an OMEGAMON Data Broker store.

The message details are for use by IBM Software Support.

System action:
The forwarder stops sending data.

User response:
Contact IBM Software Support.

KAYB0064I **Forwarder *forwarder_id*: peer closed connection, *bytes_received* out of *bytes_sent* bytes has been received**

Explanation:
Peer closed the connection before all the data has been received.

System action:
None.

User response:
On its own, this message does not indicate an error. Observe any subsequent error or warning messages for this forwarder and take the appropriate actions.

KAYB0065E **Formatting error at *site***

Explanation
OMEGAMON Data Broker failed to format a string using a standard function.

site identifies the internal location in OMEGAMON Data Broker where the error occurred.

System action:
The original string is discarded. Subsequent ZWES0101I messages report the original format string in hexadecimal.

User response:
Contact IBM Software Support.

KAYB0066W **Forwarder '*forwarder_id*' has not received an acknowledgment from its sink**

Explanation:
An OMEGAMON Data Broker forwarder has not received the expected response from its sink within the time period specified by the sink parameter RECEIVE_TIMEOUT.

System action:
The forwarder reconnects to the sink and retries sending the same data.

User response:
Check that the sink is available. If the sink is an older version that does not support version 2 of the OMEGAMON Data Broker forwarder protocol, then set the sink parameter PROTOCOL_VERSION=1 for this forwarder.

KAYB0258W **Clean-up triggered - *broker_name*/*store_name*/*subscriber_id* (*server_rc*,*broker_rc*,*service_rc*)**

Explanation
A subscriber to an OMEGAMON Data Broker store started terminating without explicitly unsubscribing. That event triggered the resource manager of the store to perform clean-up.

Typically, the subscriber is a forwarder, and the *subscriber_id* is a string of hexadecimal digits where each pair of digits represents a single-byte EBCDIC code point. Converting each pair of digits into a character produces a string with the pattern KAY.FWD.*forwarder_id*. For example, a *subscriber_id* value of D2C1E84BC6E6C44BD6D4404040404040 converts to KAY.FWD.OM, where OM is the forwarder ID.

A possible reason for this message: the Zowe cross-memory server job that runs OMEGAMON Data Broker was ended "abruptly". For example, by an MVS **CANCEL** system command.

System action:
OMEGAMON Data Broker unsubscribes the subscriber from the store.

User response
In future, to avoid this message, end the Zowe cross-memory server job by issuing an MVS **STOP** system command instead of **CANCEL**.

If this message cannot be explained by the server job being ended abruptly, then contact IBM Software Support.

KAYB0259E **Clean-up failed - *details***

Explanation:
A subscriber to an OMEGAMON Data Broker store started terminating without explicitly unsubscribing. That event triggered the resource manager of the store to perform clean-up, as reported by message KAYB0258E. The clean-up failed.

System action:
The subscriber remains subscribed to the store.

User response:
Contact IBM Software Support.

KAYBN000E **Forwarder *forwarder_id*: Unknown error: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed. The reason for the failure is unknown.

The *description*, *function*, and return code are from the point of failure, and can help identify the reason for the failure.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact IBM Software Support.

KAYBN001E **Forwarder *forwarder_id*: System error: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed while performing a POSIX system function.

The return code is from that function. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact IBM Software Support.

KAYBN002E **Forwarder *forwarder_id*: SSL/TLS error: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed while performing a GSKit SSL/TLS function.

The return code is from that function. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact IBM Software Support.

KAYBN003E **Forwarder *forwarder_id*: Not permitted: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed for one of the following reasons:

- The operation is not possible, perhaps due to temporary conditions. For example, no spare ports are currently available for network connections.
- The current user does not have permission to perform the operation.

The return code is from the function that attempted to perform the operation. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Use the *description*, *function* name, and return code to diagnose the reason for the failure.

KAYBN004E **Forwarder *forwarder_id*: Connection could not be established: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because a connection was refused or the host was unreachable.

Some common reasons:

- OMEGAMON Data Connect is not running.
- One or both of the OMEGAMON Data Broker configuration parameters for specifying the host and port on which OMEGAMON Data Connect is listening, `SINK_HOST` and `SINK_PORT`, are incorrect.

Tip: If the message includes an `errno2` value, use the z/OS command **`bpxmtext`** to display a corresponding description. For example, if the message includes `errno2=0x769F0442`, then enter:

```
bpxmtext 769F0442
```

The return code is from the function that attempted to establish the connection. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Use the *description*, *function* name, and return code to diagnose the reason for the failure.

Related reference

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

KAYBN005E **Forwarder *forwarder_id*: Operation timed out: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because an operation timed out.

The return code is from the function that attempted to perform the operation. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Consider adjusting the values of the OMEGAMON Data Broker configuration parameters for timeout and retry. Otherwise, contact your system network support.

Related reference

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

KAYBN006E **Forwarder *forwarder_id*: Connection lost: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because the connection was closed by the peer or dropped.

The return code is from the function that detected the lost connection. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact your system network support.

KAYBN007E **Forwarder *forwarder_id*: Key ring password error: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because the key ring password was missing, wrong, or expired.

The return code is from the function that detected the error. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Use the OMEGAMON Data Broker configuration parameter **STASH** or **PASSWORD** to specify the correct password.

KAYBN008E **Forwarder *forwarder_id*: Error opening key database: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because an I/O or formatting error occurred opening the key ring.

The return code is from the function that detected the error. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact your z/OS system security administrator.

KAYBN009E **Forwarder *forwarder_id*: Remote host's certificate could not be validated: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because the certificate from OMEGAMON Data Connect (the remote host in this context) could not be validated. Possible reasons include: the certificate could be self-signed, revoked, or have an unknown certificate authority (CA).

The return code is from the function that detected the error. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact your system security administrator.

KAYBN010E **Forwarder *forwarder_id*: Remote host unsupported: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

OMEGAMON Data Broker attempted to send data to OMEGAMON Data Connect, but failed because OMEGAMON Data Connect (the remote host in this context) performed an action that is not supported.

The return code is from the function that detected the error. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Contact your z/OS security administrator with the details of this message. After resolving the issue, restart the Zowe cross-memory server that is running OMEGAMON Data Broker.

KAYBN011E **Forwarder *forwarder_id*: Invalid argument: *description*: *function*: rc=*decimal_rc*(*hex_rc*)**

Explanation

An OMEGAMON Data Broker configuration parameter specified an invalid value.

The return code is from the function that detected the error. The *description* matches the return code.

System action:
OMEGAMON Data Broker tries to reconnect to the sink.

User response:
Address the error described in the message, and then restart the Zowe cross-memory server that is running OMEGAMON Data Broker.

Related reference

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters link data sources to OMEGAMON Data Connect.

KAYC: Messages from OMEGAMON Data Connect

Messages with the prefix KAYC are from OMEGAMON Data Connect.

OMEGAMON Data Connect writes messages to the STDOUT file.

KAYC0001I **Connecting to *hostname*:*port* store *store***

Explanation:
OMEGAMON Data Connect is attempting to connect to the OMEGAMON Data Broker specified by a `connect.input.cidb` configuration parameter.

System action:
None.

User response:
None required.

KAYC0002I **Connected to *hostname*:*port* store *store***

Explanation:
OMEGAMON Data Connect has successfully connected to the OMEGAMON Data Broker store specified by a `connect.input.cidb` configuration parameter.

System action:
None.

User response:
None required.

KAYC0003W **Connection to *hostname*:*port* lost. Reconnecting in *retry_interval* seconds**

Explanation:
OMEGAMON Data Connect has lost its connection to a OMEGAMON Data Broker specified by a `connect.input.cidb` configuration parameter.

System action:
OMEGAMON Data Connect waits for the specified interval, and then attempts to reconnect.

User response:
None required.

KAYC0004I **Disconnecting from *hostname*:*port* store *store***

Explanation:
OMEGAMON Data Connect is about to disconnect from the OMEGAMON Data Broker store specified by a `connect.input.cidb` configuration parameter.

System action:
None.

User response:
None required.

KAYC0005I **Disconnected from *hostname*:*port* store *store***

Explanation:
OMEGAMON Data Connect has disconnected from the OMEGAMON Data Broker store specified by a `connect.input.cidb` configuration parameter.

System action:
None.

User response:
None required.

KAYC0006E **An error occurred unsubscribing from CIDB: *details***

Explanation:
OMEGAMON Data Connect encountered an error disconnecting from the OMEGAMON Data Broker specified by a `connect.input.cidb` configuration parameter.

System action:
Depending on the details provided in the message, OMEGAMON Data Connect might not have disconnected from OMEGAMON Data Broker.

User response:
Review the details provided in this message. Review the messages in the output from the corresponding OMEGAMON Data Broker job.

KAYC0007E **An error occurred subscribing to CIDB: *details***

Explanation:
OMEGAMON Data Connect encountered an error connecting to the OMEGAMON Data Broker specified by a `connect.input.cidb` configuration parameter.

System action:
OMEGAMON Data Connect does not connect to OMEGAMON Data Broker.

User response:
Review the details provided in this message. Review the messages in the output from the corresponding OMEGAMON Data Broker job.

KAYC0008I **Creating mapping class for table *table_name***

Explanation

For the first time since starting, this instance of OMEGAMON Data Connect has received data for this table. "Mapping class" refers to code in OMEGAMON Data Connect that transforms data (such as OMEGAMON attributes) from their original proprietary binary format. Compare with KAYC0033I.

Tip: The frequency of incoming data is determined by the collection interval of the collection for this table. A long collection interval can mean a long delay before this message occurs.

System action:
None.

User response:
None required.

KAYC0009I **Starting TCP output service**

Explanation:
OMEGAMON Data Connect is starting the output service requested by a `connect.output.tcp` configuration parameter.

System action:
None.

User response:
None required.

KAYC0010I **Connecting to TCP sink: *sink_name* {host: *hostname*, port: *port*}**

Explanation:
OMEGAMON Data Connect is attempting to connect to the TCP sink specified by the `connect.output.tcp.sinks.`*`sink_name`* configuration parameter.

System action:
None.

User response:
None required.

KAYC0011I **Connected to TCP sink: *sink_name* {host: *hostname*, port: *port*}**

Explanation:
OMEGAMON Data Connect has successfully connected to the TCP sink specified by the `connect.output.tcp.sinks.`*`sink_name`* configuration parameter.

System action:
None.

User response:
None required.

KAYC0012E **Error connecting to TCP sink: *sink_name* {host: *hostname*, port: *port*}**

Explanation:
OMEGAMON Data Connect could not connect to the TCP sink specified by the `connect.output.tcp.sinks.`*`sink_name`* configuration parameter.

System action:
OMEGAMON Data Connect continues, but does not send output to that TCP sink.

User response:
Check that the destination *hostname*:*port* is listening for JSON Lines over TCP from OMEGAMON Data Connect.

KAYC0013E **Maximum allowed connection attempts (*max_connection_attempts*) exceeded, output to TCP sink *sink_name* {host: *hostname*, port: *port*} has stopped**

Explanation:
OMEGAMON Data Connect could not connect to the TCP sink specified by the `connect.output.tcp.sinks.`*`sink_name`* configuration parameter.

System action:
OMEGAMON Data Connect continues running, but does not send output to that TCP sink.

User response

1. Check that the destination is listening for JSON Lines over TCP from OMEGAMON Data Connect.
2. Consider changing the value of the OMEGAMON Data Connect configuration parameter `connect.output.tcp.sinks.`*`sink_name`*`.max-connection-attempts`.

3. Restart OMEGAMON Data Connect.

KAYC0014E I/O error writing to peer socket: *details*. Reconnection will be attempted in *retry_interval* seconds

Explanation

OMEGAMON Data Connect could not reconnect to the output TCP destination specified by a `connect.output.tcp` configuration parameter.

details describes the specific I/O error.

System action:
OMEGAMON Data Connect attempts reconnection after the specified interval.

User response

If OMEGAMON Data Connect cannot reconnect, or this issue occurs frequently:

1. Check that the destination *host*:*port* is listening for JSON Lines over TCP from OMEGAMON Data Connect.
2. Consider changing the value of the OMEGAMON Data Connect configuration parameter `connect.output.tcp.sinks.`*`sink_name`*`.retry-interval`.
3. Restart OMEGAMON Data Connect.

KAYC0015E Error creating JSON

Explanation:
OMEGAMON Data Connect encountered an error creating JSON for output.

System action:
The data is not sent to the output destination.

User response:
Review the error details following this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0016E Error instantiating mapping object

Explanation:
OMEGAMON Data Connect encountered an error while initializing the mapping code that transforms data (such as OMEGAMON attributes) from their original proprietary binary format.

System action:
OMEGAMON Data Connect continues, but does not process records for that table.

User response:
Review the error details following this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0018I Starting metrics service

Explanation:
OMEGAMON Data Connect is starting the Prometheus metrics output service requested by a `connect.output.prometheus` configuration parameter.

System action:
None.

User response:
None required.

KAYC0019W Unhandled metric type: *metric_type*

Explanation

An OMEGAMON Data Connect configuration parameter `connect.output.prometheus.mappings.products.tables.`*`table_name`*`.metrics.type` specified an unsupported Prometheus metric type:

```
metrics:
  - name: field_name
    type: metric_type # 1
```

1
The supported values of *`metric_type`* are `counter` and `gauge`.

System action:
The metric *`field_name`* is not published. Other metrics are unaffected.

User response:
Specify a supported metric type, and then restart OMEGAMON Data Connect.

KAYC0020E Error writing '*field_name*' metric

Explanation

OMEGAMON Data Connect encountered an error writing the metric to the Prometheus endpoint.

field_name is the value of a `connect.output.prometheus.mappings.products.tables.`*`table_name`*`.metrics.name` parameter in the OMEGAMON Data Connect configuration file.

System action:
The metric *field_name* is not published.

User response:
Review the error details following this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0021E Reflection error accessing label

Explanation:

There is a problem in the OMEGAMON Data Connect configuration file with a parameter for a Prometheus metric.

System action:
Depends on the specific issue described in the details that follow this message.

User response:
Review the error details following this message. Examine the corresponding OMEGAMON Data Connect configuration parameters under the `connect.output.prometheus` key. If you cannot resolve the issue, contact IBM Software Support.

KAYC0022I Starting Kafka input service

Explanation:
OMEGAMON Data Connect is starting the Apache Kafka input service requested by a `connect.input.kafka` configuration parameter.

System action:
None.

User response:
None required.

KAYC0023I Starting TCP input service listening on *hostname*:*port*

Explanation:
OMEGAMON Data Connect is starting the TCP input service requested by a `connect.input.tcp` configuration parameter.

System action:
None.

User response:
None required.

KAYC0024I Starting STDOUT output service

Explanation:
OMEGAMON Data Connect is starting the output service requested by a `connect.output.stdout` configuration parameter.

System action:
None.

User response:
None required.

KAYC0025I Starting Kafka output service

Explanation:
OMEGAMON Data Connect is starting the Apache Kafka output service requested by a `connect.output.kafka` configuration parameter.

System action:
None.

User response:
None required.

KAYC0026I Creating JSON mapping provider

Explanation:
OMEGAMON Data Connect is initializing the code that maps data (such as OMEGAMON attributes) from their original proprietary data format to JSON.

System action:
None.

User response:
None required.

KAYC0027I Stopping TCP listener

Explanation:
OMEGAMON Data Connect is stopping the TCP listener.

System action:
None.

User response:
None required.

KAYC0028I Source *hostname*:*port* has connected

Explanation:
OMEGAMON Data Broker, at the specified *hostname* and *port*, has connected to OMEGAMON Data Connect.

System action:
None.

User response:
None required.

KAYC0029I Source *hostname*:*port* has disconnected

Explanation:
OMEGAMON Data Broker, at the specified *hostname* and *port*, has disconnected from OMEGAMON Data Connect.

System action:
None.

User response:
None required.

KAYC0031W Event publication error

Explanation

OMEGAMON Data Connect encountered an error publishing an event.

Flood-controlled: To avoid duplicate messages flooding the log, this message is subject to flood control. OMEGAMON Data Connect might suppress duplicate messages within the flood control interval. For details, see the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action:

Depends on the details in the KAYC0073I message that follows this message.

User response:
See the details in the KAYC0073I message that follows this message. If you cannot resolve the issue, contact IBM Software Support.

Related reference

Connect-specific logging parameters
OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

KAYC0032I **Stopping TCP output service**

Explanation:
OMEGAMON Data Connect is stopping its TCP output service.

System action:
None.

User response:
None required.

KAYC0033I **Table *table_name* received from *origin_type origin_name***

Explanation

For the first time since starting, this instance of OMEGAMON Data Connect has received data for this table from this *origin_name*.

origin_type and *origin_name* depend on the table. Examples:

origin_type	***origin_name***
`Sysplex`	The name of the sysplex
`CICS Region`	The name of the CICS region

Compare with KAYC0008I.

System action:
None.

User response:
None required.

KAYC0034I **Stopping server**

Explanation:
OMEGAMON Data Connect is stopping.

System action:
None.

User response:
None required.

KAYC0035I **Build: *build_identifier***

Explanation:
Identifies the OMEGAMON Data Connect build. This identifier is for use by IBM Software Support.

System action:
None.

User response:
None required.

KAYC0036I ***filter_scope* filter selected table: *table_name*, fields: *field_list***

Explanation

The OMEGAMON Data Connect filter for JSON-format outputs has been configured to select only the specified fields from this table for processing.

The value of *field_list* depends on the filter:

- If the filter specifies a list of fields, then *field_list* is a comma-separated list of field names enclosed in square brackets:

  ```
  [field_name, field_name, ...]
  ```

- If the filter does not specify a list of fields, then all fields in the table are selected, and *field_list* has the value `ALL`

filter_scope identifies the scope of the filter:

- `Kafka`
- `STDOUT`
- `TCP sink: sink_name`
- `HTTP endpoint: endpoint_name`

System action:
None.

User response:
None required.

Related reference

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

KAYC0037I **Registered metric for table: *table_name*, field: *field_name*, type: *metric_type*, labels: *label_list***

Explanation:
OMEGAMON Data Connect has been configured to output a Prometheus metric with these details.

System action:
None.

User response:
None required.

Related reference

Prometheus output parameters

OMEGAMON Data Connect can publish attributes to a Prometheus endpoint. OMEGAMON Data Connect Prometheus output parameters describe the Prometheus endpoint and which attributes to publish.

KAYC0038I Starting console listener

Explanation

OMEGAMON Data Connect is listening for commands from the console.

For example, if OMEGAMON Data Connect is running on z/OS, OMEGAMON Data Connect is listening for MVS system **MODIFY** commands.

System action:
None.

User response:
None required.

KAYC0039W Invalid modify command: *command*

Explanation

OMEGAMON Data Connect received an invalid command from the console.

For example, if OMEGAMON Data Connect is running on z/OS, OMEGAMON Data Connect received an invalid MVS system **MODIFY** command.

System action:
The command is ignored.

User response:
Enter a valid console command; on z/OS, a valid MVS system **MODIFY** command.

KAYC0040E Error creating socket

Explanation:
OMEGAMON Data Connect encountered an error creating a socket network connection for TCP output.

System action:
OMEGAMON Data Connect does not send data to the TCP output destination.

User response:
Review the error details following this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0041E Error creating SSL context

Explanation:
OMEGAMON Data Connect encountered an error creating a secure (SSL/TLS) socket network connection for TCP input from OMEGAMON Data Broker.

System action:
OMEGAMON Data Connect does not receive data from OMEGAMON Data Broker.

User response:
Review the error details following this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0042I Starting TCP sink: *sink_name* {host: *hostname*, port: *port*}

Explanation:
OMEGAMON Data Connect is starting a thread for the TCP sink specified by the `connect.output.tcp.sinks.`*`sink_name`* configuration parameter.

System action:
None.

User response:
None required.

KAYC0043I Stopping TCP sink: *sink_name* {host: *hostname*, port: *port*}

Explanation:
OMEGAMON Data Connect is stopping the thread for the TCP sink specified by the `connect.output.tcp.sinks.`*`sink_name`* configuration parameter.

System action:
None.

User response:
None required.

KAYC0044I Event publication for table *table_name* has been disabled

Explanation:
OMEGAMON Data Connect configuration parameters have disabled publication of data for this table.

System action:
OMEGAMON Data Connect does not publish data for this table.

User response:
None required.

KAYC0045E Field '*field_name*' does not exist in table '*table_name*', product '*product_code*'

Explanation:
The OMEGAMON Data Connect configuration parameters refer to a field that does not exist in the specified table.

System action:
OMEGAMON Data Connect stops.

User response

1. Check that the field exists and that you have spelled the field name correctly, in the correct case.
2. Edit the configuration parameters, and then restart OMEGAMON Data Connect.
3. If you cannot resolve the issue, contact IBM Software Support.

Related reference

OMEGAMON attribute dictionary
OMEGAMON Data Connect includes a dictionary of OMEGAMON attributes in a set of YAML files.

KAYC0046E **Table '*table_name*' does not exist in product '*product_code*'**

Explanation:
The OMEGAMON Data Connect configuration parameters refer to a table that does not exist in the specified product.

System action:
OMEGAMON Data Connect stops.

User response

1. Edit the configuration parameters.
2. Restart OMEGAMON Data Connect.
3. If you cannot resolve the issue, contact IBM Software Support.

KAYC0047I **Starting broker stats service**

Explanation:
OMEGAMON Data Broker sends statistics about its activity to OMEGAMON Data Connect. OMEGAMON Data Connect publishes these OMEGAMON Data Broker statistics to the Spring Boot Actuator Prometheus endpoint `/actuator/prometheus` as metrics with the prefix `odp_broker`.

System action:
None.

User response:
None required.

KAYC0049E **Error publishing to Kafka**

Explanation:
OMEGAMON Data Connect encountered an error attempting to send data to Kafka.

System action

OMEGAMON Data Connect performs the following actions:

1. Flushes (discards) any unsent data queued for output to Kafka
2. Stops sending data to Kafka.
3. Attempts to reconnect to Kafka.

   If the reconnection attempt succeeds, then OMEGAMON Data Connect restarts sending data to Kafka. However, the previously flushed data is lost.
4. If the reconnection attempt fails, then OMEGAMON Data Connect reports error message KAYC0050E, and permanently stops sending data to Kafka.

User response

1. Check that you have specified the correct *`host:port`* connection details for the Kafka servers.
2. Consider the values that you have set for the Kafka output parameters `retry-interval` and `max-connection-attempts`.
3. Investigate the Kafka log for potential causes of the error.

KAYC0050E **Kafka output service has stopped**

Explanation:
This message follows KAYC0049E, which reports an error sending data to Kafka. This message reports that OMEGAMON Data Connect was unable to connect to Kafka after that error.

System action:
OMEGAMON Data Connect permanently stops sending data to Kafka.

User response:
See the response for KAYC0049E.

KAYC0051E **Error connecting to Kafka. Retrying in *retry-interval* seconds**

Explanation:
OMEGAMON Data Connect attempted but failed to connect to Kafka.

System action:
OMEGAMON Data Connect will retry connecting to Kafka after the number of seconds specified by the Kafka output parameter `retry-interval`. The maximum number of attempts is determined by the parameter `max-connection-attempts`.

User response:
None required.

KAYC0053E **Nested filter includes are not supported**

Explanation:
OMEGAMON Data Connect found an `include` parameter in a filter include file.

System action:
OMEGAMON Data Connect stops.

User response

1. Remove the `include` parameter from the filter include file.
2. Restart OMEGAMON Data Connect.

KAYC0054E **Filter include file *file_path* was not found**

Explanation:
OMEGAMON Data Connect could not find the filter include file, specified by an `include` parameter, either in the file system or in the class path.

System action:
OMEGAMON Data Connect stops.

User response:
Edit the `include` parameter to point to the correct file path. Restart OMEGAMON Data Connect.

KAYC0056I **Table *table_name* has been disabled for outputs that use this filter**

Explanation:
This message follows KAYC0057W, which reports an error in a filter condition expression. As a result of that error, OMEGAMON Data Connect disables the table for outputs that use this filter.

System action:
OMEGAMON Data Connect continues processing. However, OMEGAMON Data Connect stops processing records of this table for outputs that use this filter.

User response:
No response required for this message. See the response for message KAYC0057W.

Related reference

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

KAYC0057W **Filter condition for *product_code.table_name* failed, expression '*expression*'**

Explanation

OMEGAMON Data Connect encountered an error while evaluating the specified filter condition *expression*. The Spring Expression Language (SpEL) expression caused a runtime exception.

Typical causes include:

Divide-by-zero error
The expression uses an integer field as the denominator in a division operation. In this case, the field value is expected to always be nonzero. If the field value is zero, a divide-by-zero error occurs.

Null field value
A null value can cause an error, depending on where that value occurs in an expression. For details on how SpEL treats null field values, see the Spring documentation.

Misspelled field name
The field will not be found.

Attempting to set the value of a read-only field
For example, mistakenly using a single equal sign (=) to compare for equality instead of the correct two consecutive equal signs (==).

Flood-controlled: To avoid duplicate messages flooding the log, this message is subject to flood control. OMEGAMON Data Connect might suppress duplicate messages within the flood control interval. For details, see the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action

OMEGAMON Data Connect continues processing as if the expression had successfully returned a false value. The corresponding input record is discarded.

If the OMEGAMON Data Connect configuration parameter `disable-table-on-error` is `true` for this filter, then OMEGAMON Data Connect performs the following actions:

1. Stop processing records that use the expression; disable the table for outputs that use this filter.

   If the expression is in an output-level filter, then OMEGAMON Data Connect disables the table for that output only. If the expression is in a global-level filter, then OMEGAMON Data Connect disables the table for *all* outputs that use the global-level filter.
2. Report information message KAYC0056I.

User response:
See the details in the KAYC0072I message that follows this message. Consider adjusting the expression in the OMEGAMON Data Connect configuration to avoid triggering the runtime exception.

Related reference

Filters for JSON-format outputs
You can optionally filter the data to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, HTTP, Kafka, and STDOUT.

Connect-specific logging parameters

OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

KAYC0058W **Duplicate field name '*field_name*' in the mapping class for *product_code.table_name table_version***

Explanation

A mapping class in an OMEGAMON Data Connect mapping extension JAR file contains more than one definition for the same *field_name*.

The *table_version* is for use by IBM Software Support. The *table_version* is specific to the table; it is not related to the version of the mapping extension JAR file that contains the table schema.

System action:
OMEGAMON Data Connect ignores the duplicate field definition and continues processing.

User response:
Contact IBM Software Support.

KAYC0059W **No fields found in the mapping class for *product_code.table_name table_version***

Explanation

A mapping class in an OMEGAMON Data Connect mapping extension JAR file contains no field definitions for the specified table.

The *table_version* is for use by IBM Software Support. The *table_version* is specific to the table; it is not related to the version of the mapping extension JAR file that contains the table schema.

System action:
OMEGAMON Data Connect does not process any fields in this table (for example, OMEGAMON attributes in this group).

User response:
Contact IBM Software Support.

KAYC0060W **Unsupported specification version *mapping_extension_framework_version* in *class_name* (*jar_file_path*)**

Explanation

The OMEGAMON Data Connect mapping extension identified by *class_name* and *jar_file_path* is not supported by the running version of OMEGAMON Data Connect.

OMEGAMON Data Connect involves one JAR file containing core classes and multiple JAR files containing mapping extension classes. The mapping extension classes must be compatible with the core classes. Compatibility is determined by the versions of the OMEGAMON Data Connect framework that were used to build the JAR files:

mapping_extension_framework_version
(Reported in this message.) The version of the OMEGAMON Data Connect framework that was used to build the mapping extension JAR file.

This is the value of the `Specification-Version` header in the manifest in the mapping extension JAR file.

core_framework_version
(Reported in message KAYC0067I.) The version of the OMEGAMON Data Connect framework that was used to build the OMEGAMON Data Connect core JAR file, `odp-server-`*`version`*`.jar`.

Given version numbers in the three-part format *`major.minor.patch`*, the *mapping_extension_framework_version* must meet both of the following requirements:

- The *mapping_extension_framework_version* must have the same *major* version as *core_framework_version*.
- The *mapping_extension_framework_version* must have the same or earlier *minor.patch* version as *core_framework_version*.

For example, given a *core_framework_version* value of 2.3.1:

- Supported values of *mapping_extension_framework_version* include: 2.3.1, 2.3.0, 2.1.0
- Unsupported values of *mapping_extension_framework_version* include: 3.1.0, 2.4.0, 2.3.2, 1.1.0

System action

OMEGAMON Data Connect ignores the mapping extension and continues processing.

Mapping extensions contain mapping classes that enable OMEGAMON Data Connect to process different types of incoming record. OMEGAMON Data Connect discards records for which there is no mapping class.

User response:
Check that the OMEGAMON Data Connect runtime option `-Dodp.ext` refers to the correct set of mapping extensions for this version of OMEGAMON Data Connect. If you cannot resolve the issue, contact IBM Software Support.

Related information

KAYC0067I

Framework version = *connect_framework_version*

KAYC0061W Malformed record received

Explanation

OMEGAMON Data Connect received a malformed record from OMEGAMON Data Broker. The record does not have the structure that OMEGAMON Data Connect expected.

Flood-controlled: To avoid duplicate messages flooding the log, this message is subject to flood control. OMEGAMON Data Connect might suppress duplicate messages within the flood control interval. For details, see the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action:
OMEGAMON Data Connect discards the record and continues processing.

User response:
See the details in the KAYC0071I message that follows this message. If you cannot resolve the issue, contact IBM Software Support.

Related reference

Connect-specific logging parameters
OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

KAYC0062W Mapping class not found for *product_code.table_name table_version*

Explanation

OMEGAMON Data Connect received a record from OMEGAMON Data Broker with a combination of *product_code*, *table_name*, and *table_version* that is not supported by any of the available mapping classes.

The *table_version* is for use by IBM Software Support. The *table_version* is specific to the table; it is not related to the version of the mapping extension JAR file that contains the table schema.

Flood-controlled: To avoid duplicate messages flooding the log, this message is subject to flood control. OMEGAMON Data Connect might suppress duplicate messages within the flood control interval. For details, see the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action:
OMEGAMON Data Connect discards the record and continues processing.

User response:
Check that the paths specified by the OMEGAMON Data Connect runtime option `-Dodp.ext` refer to all of the available mapping extension JAR files. If you cannot resolve the issue, contact IBM Software Support.

Related reference

Connect-specific logging parameters
OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

OMEGAMON attribute dictionary
OMEGAMON Data Connect includes a dictionary of OMEGAMON attributes in a set of YAML files.

OMEGAMON monitoring agents supported by OMEGAMON Data Provider
OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

KAYC0063I Internal Statistics:

Explanation:
This message is followed by a dump of internal statistics from OMEGAMON Data Connect.

System action:
None.

User response:
None required.

KAYC0064W Specification version is missing in *class_name* (*jar_file_path*)

Explanation

The OMEGAMON Data Connect mapping extension identified by *class_name* and *jar_file_path* has no `Specification-Version` header in its JAR file manifest.

Mapping extensions must include a specification version. The specification version indicates the version of the OMEGAMON Data Connect framework that was used to build the mapping extension.

System action

OMEGAMON Data Connect ignores the mapping extension and continues processing.

Mapping extensions contain mapping classes that enable OMEGAMON Data Connect to process different types of incoming record. OMEGAMON Data Connect discards records for which there is no mapping class.

User response:
Contact IBM Software Support.

KAYC0065W Implementation version is missing in *class_name* (*jar_file_path*)

Explanation
The OMEGAMON Data Connect mapping extension identified by *class_name* and *jar_file_path* has no `Implementation-Version` header in its JAR file manifest.

Mapping extensions must include an implementation version. The implementation version indicates the version of the mapping extension.

System action
OMEGAMON Data Connect ignores the mapping extension and continues processing.

Mapping extensions contain mapping classes that enable OMEGAMON Data Connect to process different types of incoming record. OMEGAMON Data Connect discards records for which there is no mapping class.

User response:
Contact IBM Software Support.

KAYC0066W **Duplicate mapping class for *product_code.table_name table_version* found (impl *duplicate_package_impl_version* in *duplicate_jar_file_path* vs *existing_package_impl_version* in *existing_jar_file_path*)**

Explanation:
OMEGAMON Data Connect has found a mapping class for the same table, with the same table schema version, in two mapping extension JAR files.

System action
OMEGAMON Data Connect uses the mapping class from the mapping extension JAR file with the most recent implementation version.

That is, if the implementation version of the newly found (duplicate) mapping extension JAR file is later than the implementation version of the previously found (existing) mapping extension JAR file, then OMEGAMON Data Connect uses the mapping class in the newly found file. Otherwise, OMEGAMON Data Connect ignores the mapping class in the newly found file.

User response
Investigate why both of these JAR files exist in the paths specified by the OMEGAMON Data Connect runtime option `-Dodp.ext`.

For example, it's possible that you might deliberately have stored in your *user* directory a newer version of a JAR file that also exists in the *installation* directory; in which case, this warning is expected, and you can ignore it. However, if the presence of the duplicate is unexpected, then you need to review how you manage these JAR files.

If you cannot resolve the issue, contact IBM Software Support.

KAYC0067I **Framework version = *connect_framework_version***

Explanation
Identifies the version of the OMEGAMON Data Connect framework that was used to build the OMEGAMON Data Connect core JAR file, `odp-server-`*`version`*`.jar`.

connect_framework_version is the value of the `Specification-Version` header in the manifest in the core JAR file.

System action:
None.

User response:
None required.

Related information

KAYC0060W
Unsupported specification version *mapping_extension_framework_version* in *class_name* (*jar_file_path*)

KAYC0068I **Extension file *jar_file_path* found (spec=*framework_version*, impl=*mapping_extension_version*)**

Explanation
OMEGAMON Data Connect reports this message for each mapping extension JAR file that it finds in the directories that are listed in the OMEGAMON Data Connect runtime option `-Dodp.ext`.

framework_version
The version of the OMEGAMON Data Connect framework that was used to build the mapping extension JAR file.

This is the value of the `Specification-Version` header in the manifest in the mapping extension JAR file.

mapping_extension_version
The version of the mapping extension.

This is the value of the `Implementation-Version` header in the manifest in the mapping extension JAR file.

System action:
None.

User response:
None required.

KAYC0069I ***class_count*** **mapping classes found in** ***jar_file_path***

Explanation:
OMEGAMON Data Connect reports the number of mapping classes it finds in each mapping extension JAR file.

System action:
None.

User response:
None required.

KAYC0070W **Failed to read JAR file** ***jar_file_path*****:** ***details***

Explanation:
OMEGAMON Data Connect could not read the specified mapping extension JAR file.

System action

OMEGAMON Data Connect ignores the mapping extension and continues processing.

Mapping extensions contain mapping classes that enable OMEGAMON Data Connect to process different types of incoming record. OMEGAMON Data Connect discards records for which there is no mapping class.

User response:
Review the details provided by this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0071I **Malformed record reason:** ***details***

Explanation:
This message reports the reason for the preceding KAYC0061W message.

System action:
None.

User response:
Review the details provided by this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0072I **Filter failure reason:** ***details***

Explanation:
This message reports the reason for the preceding KAYC0057W message.

System action:
None.

User response:
Review the details provided by this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0073I **Event publication error reason:** ***details***

Explanation:
This message reports the reason for the preceding KAYC0031W message.

System action:
None.

User response:
Review the details provided by this message. If you cannot resolve the issue, contact IBM Software Support.

KAYC0074I ***total_messages_suppressed*** **messages have been suppressed in** ***logger_name*** **in the last** ***flood_control_interval*** **seconds:**

Explanation

Reports the number of messages that OMEGAMON Data Connect has suppressed from the log in the last flood control interval.

OMEGAMON Data Connect reports this message only if messages have been suppressed.

The *logger_name* is for use by IBM Software Support.

To configure flood control, set the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action:
This message is followed by a report of the suppressed messages, sorted in order of highest to lowest number of messages suppressed.

User response:
None required.

Related reference

Connect-specific logging parameters
OMEGAMON Data Connect has its own specific logging parameters, separate from the common Spring Boot logging properties.

KAYC0075W **Only JAR files are supported. File** ***file_path*** **was ignored**

Explanation

The OMEGAMON Data Connect runtime option `-Dodp.ext` specified a file path that does not end with the file extension `.jar`.

`-Dodp.ext` specifies the locations of mapping extension JAR files. File paths specified in `-Dodp.ext` must end with the file extension `.jar`.

System action:
OMEGAMON Data Connect ignores the file.

User response:
Correct or remove the file path in the value of `-Dodp.ext`.

KAYC0076I **Starting HTTP output service**

Explanation:
OMEGAMON Data Connect is starting the output service requested by a `connect.output.http` configuration parameter.

System action:
None.

User response:
None required.

KAYC0077I **Stopping HTTP output service**

Explanation:
OMEGAMON Data Connect is stopping its HTTP output service.

System action:
None.

User response:
None required.

KAYC0078I **Starting output to HTTP endpoint: *endpoint_name* {url: *url*}**

Explanation:
OMEGAMON Data Connect is starting a thread for the HTTP output specified by the `connect.output.http.endpoints.`*`endpoint_name`* configuration parameter.

System action:
None.

User response:
None required.

KAYC0079E **Error sending data to HTTP endpoint *endpoint_name*: *details***

Explanation

OMEGAMON Data Connect encountered a problem sending data to the HTTP endpoint specified by the `connect.output.http.endpoints.`*`endpoint_name`* configuration parameter.

Flood-controlled: To avoid duplicate messages flooding the log, this message is subject to flood control. OMEGAMON Data Connect might suppress duplicate messages within the flood control interval. For details, see the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action:
OMEGAMON Data Connect continues. However, data in a failed POST request to the endpoint is lost.

User response:
Check that the HTTP endpoint is listening for POST requests from OMEGAMON Data Connect.

KAYC0080E **Maximum allowed failures (*max_failures*) exceeded, output to HTTP endpoint *endpoint_name* has stopped**

Explanation

The number of failures attempting to send a request to this HTTP endpoint has exceeded the maximum number of allowed failures specified by the OMEGAMON Data Connect configuration parameter `connect.output.http.endpoints.`*`endpoint_name`*`.max-failures`.

Possible failures include timeouts, connection failures, and unsuccessful responses.

System action:
OMEGAMON Data Connect continues running, but does not send any more requests to this HTTP endpoint.

User response

1. Check that the HTTP endpoint is listening for POST requests from OMEGAMON Data Connect.
2. Consider changing the value of the `max-failures` parameter.
3. Restart OMEGAMON Data Connect.

KAYC0081I **Stopping output to HTTP endpoint: *endpoint_name* {url: *url*}**

Explanation:
OMEGAMON Data Connect is stopping the thread for the HTTP endpoint specified by the `connect.output.http.endpoints.`*`endpoint_name`* configuration parameter.

System action:
None.

User response:
None required.

KAYC0082E **Error creating HTTP client for endpoint *endpoint_name*.**

Explanation:
OMEGAMON Data Connect could not create an HTTP client for the endpoint specified by the `connect.output.http.endpoints.`*`endpoint_name`* configuration parameter.

System action:
OMEGAMON Data Connect stops.

User response

1. Review the error details following this message.

2. Check that the HTTP endpoint is listening for POST requests from OMEGAMON Data Connect.
3. Review the configuration of `connect.output.http.endpoints.`*`endpoint_name`*.
4. Restart OMEGAMON Data Connect.
5. If you cannot resolve the issue, contact IBM Software Support.

KAYC0083E **Fatal error, output to HTTP endpoint *endpoint_name* has stopped**

Explanation:
OMEGAMON Data Connect encountered an unrecoverable error attempting to send to the HTTP endpoint specified by the `connect.output.http.endpoints.`*`endpoint_name`* configuration parameter.

System action:
OMEGAMON Data Connect continues running, but does not send any more requests to that HTTP endpoint.

User response

1. Check that the HTTP endpoint is listening for POST requests from OMEGAMON Data Connect.
2. Review the configuration of `connect.output.http.endpoints.`*`endpoint_name`*.
3. Restart OMEGAMON Data Connect.

KAYC0084W **Queue capacity (*queue_capacity*) reached for output *output_description***

Explanation

The OMEGAMON Data Connect internal queue for this output has reached the maximum number of records specified by the `connect.event-publisher.queue-capacity` configuration parameter.

Typical reasons are related to high data volume, such as high data frequency. For example:

- The output destination has stopped reading incoming records because it has run out of space.
- Incoming records are arriving faster than OMEGAMON Data Connect can process them.

Flood-controlled: To avoid duplicate messages flooding the log, this message is subject to flood control. OMEGAMON Data Connect might suppress duplicate messages within the flood control interval. For details, see the OMEGAMON Data Connect configuration parameters under `connect.logging.flood-control`.

System action

1. The queue rejects (drops) any new incoming records until the queue falls within capacity. This output loses data.
2. OMEGAMON Data Connect continues to read incoming records, even if all queues are at capacity.
3. OMEGAMON Data Connect continues to report this message until the queue falls within capacity.

Other outputs are unaffected because each output has its own queue. For example, suppose you have configured multiple outputs, and the destination of one of those outputs stops reading records. The queue for that output grows until it reaches capacity, and then rejects any new incoming records. Meanwhile, the queues for other outputs continue accepting incoming records.

User response

1. Check the health of the output destination.

   For example, check the log of the destination analytics platform for the following messages:

   - Errors such as "No space left on device"
   - Warnings about data arriving faster that it can be processed, including internal queues or buffers reaching capacity
2. If the destination is healthy, review the following aspects of your OMEGAMON Data Provider environment:

   - The value of `connect.event-publisher.queue-capacity`.

     Consider whether you need to increase the value to match your site-specific conditions.
   - The data frequency of records that you are sending to OMEGAMON Data Connect.

     For example, review your OMEGAMON collection intervals. Consider whether you can reduce the frequency of some collection intervals.
   - The processing resources and performance characteristics of the system on which you are running OMEGAMON Data Connect.

     Consider whether you need faster hardware.
   - The topology of your OMEGAMON Data Provider environment.

     Consider whether to divide the workload across more instances of OMEGAMON Data Connect.

Product legal notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.