IBM Z OMEGAMON Data Provider
Version 1.1

Installation and User's Guide

Note:

Before using this information and the product it supports, read “Product legal notices” on page 125.

2022-09-29 edition

This edition applies to IBM Z® OMEGAMON® Data Provider Version 1.1 with the PTF for APAR OA63539, and to all subsequent releases and modifications until otherwise indicated in new editions.

Figures

1. OMEGAMON Data Provider makes attributes available outside of OMEGAMON..... 5

2. Components involved in OMEGAMON Data Provider..... 6

3. Example OMEGAMON Data Provider topology..... 8

4. Example topology with separate instances of OMEGAMON Data Connect for production and development..... 9

5. Choice of destinations for attributes..... 10

6. OMEGAMON Data Provider communication protocols with or without TLS..... 11

7. OMEGAMON Data Provider configurable parts..... 18

8. Architecture of the getting started exercise: sending JSON Lines over TCP..... 19

9. Excerpt of example OMEGAMON Data Provider collection configuration member, KAYOPEN..... 20

10. JCL procedure that starts the Zowe cross-memory server, PROCLIB(ZWESIS01)..... 22

11. Instana ingests attributes from OMEGAMON Data Provider as JSON Lines over TCP..... 25

12. Configuring which attributes OMEGAMON Data Provider sends, and to where..... 27

13. Elasticsearch index template that maps string fields to the keyword data type..... 30

14. Logstash pipeline configuration to ingest JSON Lines over TCP from OMEGAMON Data Connect..... 31

15. OMEGAMON Data Provider configuration points: Collection, Broker, Connect..... 39

16. OMEGAMON Data Provider collection configuration parameters control where attributes are sent..... 40

17. OMEGAMON Data Broker configuration points: store, forwarder, and output (sink)..... 46

18. OMEGAMON Data Broker configuration: one store, one or more forwarders..... 46

19. OMEGAMON Data Connect configuration points: input from OMEGAMON Data Broker and various outputs..... 55

20. OMEGAMON Data Connect configuration: TCP input..... 57

21. OMEGAMON Data Connect configuration: TCP output..... 61

22. OMEGAMON Data Connect configuration: Kafka output..... 67

23. OMEGAMON Data Connect configuration: Prometheus output from an HTTP(S) perspective..... 70

24. OMEGAMON Data Provider is a Prometheus target..... 70

25. Example Prometheus text-format output..... 73

Tables

1. Connections between OMEGAMON Data Provider components, with links to security parameter descriptions..... 11

2. Historical data collections for Instana: z/OS..... 26

3. Historical data collections for Instana: CICS..... 26

4. Historical data collections for Instana: Db2..... 26

5. OMEGAMON Data Provider configuration points, configuration members, and sample members..... 39

6. Monitoring agents supported by OMEGAMON Data Provider, with links to attributes documentation..117

Contents

Figures iii

Tables v

About this document ix

What's new 1

Introduction 5
- Architecture 6
- Topology 7
- Attribute destinations 10
- Security 11
- Starter dashboards 12
- Prerequisites 12

Installing 15

Overview of configurable parts 17

Getting started 19
- Configuring which collections to send 19
- Configuring OMEGAMON Data Broker 21
- Configuring OMEGAMON Data Connect 23
- Integrating analytics platforms 25
  - Instana 25
  - Elastic Stack 29
  - Splunk 31
- Starting OMEGAMON Data Provider 33

Modifying running components 35
- Reloading collection configuration 35
- Displaying OMEGAMON Data Broker status 35
- Restarting OMEGAMON Data Connect 36
- Stopping components on z/OS 36

Adding more collections 37

Configuration parameters 39
- Collection 40
- OMEGAMON Data Broker 45
- OMEGAMON Data Connect 55
  - TCP input 57
  - TCP output 61
  - Kafka output 67
  - Prometheus output 69
  - STDOUT output 73
  - Filters for JSON-format outputs 74

Event publisher....83
Server....83
Logging....86

Troubleshooting....89
Gathering diagnostic information....89
Common issues....90
OMEGAMON Data Connect fails with `charset.MalformedInputException`....90
No KPQH037I or KPQH038I message for a table....91

Messages....93
Expected messages....94
KAYL, KPQD, KPQH: Messages from OMEGAMON collection tasks....98
KAYB: Messages from OMEGAMON Data Broker....101
KAYC: Messages from OMEGAMON Data Connect....107

Reference....117
Supported monitoring agents....117
Attribute dictionary....118
Attribute names versus field names....119
Attribute groups versus table names....121
Fields introduced by OMEGAMON Data Connect....122
JSON output characteristics....123

Product legal notices....125

About this document

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

What's new in OMEGAMON Data Provider

A summarized history of significant updates.

September 2022: APAR OA63539

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

Documentation updates:

"Attribute destinations" on page 10
A new topic about choosing the destinations of collected attributes.

"Expected messages" on page 94
A new topic that lists the normal messages that you should expect from each component involved in OMEGAMON Data Provider.

"OMEGAMON Data Provider collection configuration parameters" on page 40

- 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 collections to OMEGAMON Data Provider" on page 37
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 7.

For configuration details, see "OMEGAMON Data Broker configuration parameters" on page 45.

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 more monitoring agents

- MQ
  - IBM OMEGAMON for Messaging on z/OS, V7.5
- Networks

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

Support for Instana

IBM Observability by Instana Application Performance Monitoring 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 updates

- 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 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 updates

- 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 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 Tivoli OMEGAMON XE 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.

Introduction to OMEGAMON Data Provider

OMEGAMON Data Provider makes OMEGAMON attributes available to applications and analytics platforms outside of OMEGAMON.

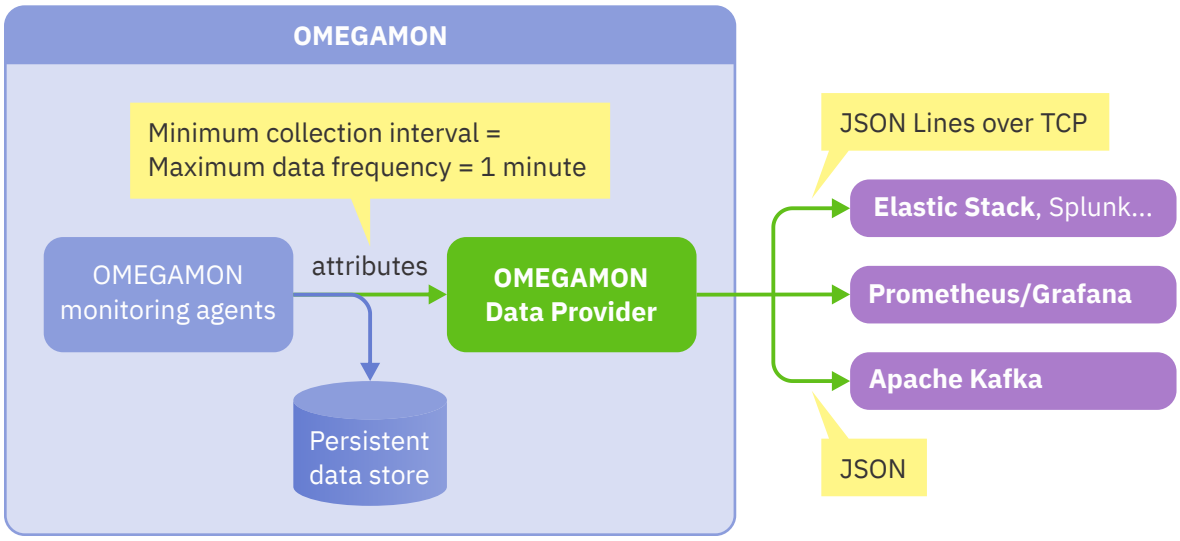

Figure 1. OMEGAMON Data Provider makes attributes available outside of OMEGAMON

OMEGAMON attributes

OMEGAMON agents monitor performance, behavior, and resource usage metrics of systems and applications on z/OS. OMEGAMON refers to these metrics as *attributes*.

Related attributes are organized into *attribute groups*, also referred to as *tables*. To control which attribute groups OMEGAMON collects and how frequently it collects them, you create *historical collections*. To control the frequency of collection for a group, you specify a *collection interval*: a minimum of 1 minute to a maximum of 1 day.

To create historical collections, use the OMEGAMON enhanced 3270 user interface (e3270UI) or 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). For longer-term storage, you can also store attributes in Tivoli Data Warehouse.

Making attributes available outside of OMEGAMON

OMEGAMON Data Provider introduces the following output methods for collected attributes:

- JSON Lines over TCP
- Prometheus endpoints
- JSON in Apache Kafka topics

These output methods are designed to be easily ingested by applications and analytics platforms outside of OMEGAMON.

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 to an analytics platform such as the Elastic Stack or Splunk, then the analytics platform receives data for that attribute group every minute.

Related reference

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

OMEGAMON Data Provider architecture

OMEGAMON Data Provider extends OMEGAMON collection tasks and introduces two components: OMEGAMON Data Broker and OMEGAMON Data Connect.

OMEGAMON Data Provider does not affect any existing historical data collection. Rather, OMEGAMON Data Provider offers a new destination for collected attributes, as an alternative to, or in addition to, the OMEGAMON persistent data store (PDS).

The following figure shows the components involved in OMEGAMON Data Provider:

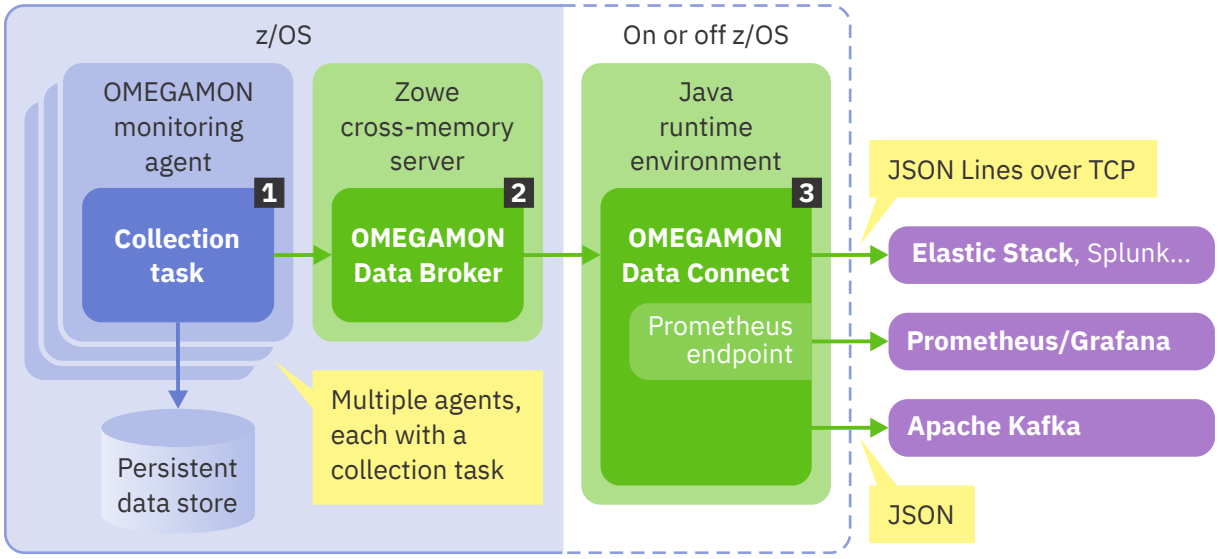

Figure 2. Components involved in OMEGAMON Data Provider

1 Collection task

OMEGAMON Data Provider extends the existing OMEGAMON collection task in two ways:

- The collection task looks for the member RKANPARU(KAYOPEN).

  If this member does not exist, then OMEGAMON Data Provider is dormant and collected attributes will be stored in PDS only.

  Otherwise, this member is a YAML document that specifies the destinations of attributes according to their table and collection interval. The destinations are: PDS, OMEGAMON Data Provider, both, or neither.

  Sending attributes to OMEGAMON Data Provider only, not PDS, is sometimes referred to as *passthrough*.

- The collection task sends attributes to OMEGAMON Data Broker.

2 OMEGAMON Data Broker

OMEGAMON Data Broker receives attributes from OMEGAMON collection tasks and forwards the attributes over a TCP network to OMEGAMON Data Connect.

OMEGAMON Data Broker is a plugin 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 in a single load module that has no dependencies on other Zowe components. For details, see "Prerequisites for OMEGAMON Data Provider" on page 12.

The Zowe cross-memory server runs in its own z/OS address space on the same z/OS instance as the OMEGAMON collection tasks from which it receives attributes.

The behavior of OMEGAMON Data Broker is determined by configuration parameters in the Zowe cross-memory server configuration member, PARMLIB(ZWESIP*xx*), consisting of plain-text key-value pairs.

3 OMEGAMON Data Connect

OMEGAMON Data Connect receives attributes from OMEGAMON Data Broker, transforms the attributes from their proprietary binary data format, and publishes the attributes using the following methods:

- JSON Lines over TCP
- Prometheus endpoints
- JSON in Apache Kafka topics

Each instance of OMEGAMON Data Connect can publish to multiple destinations: one Prometheus output, one Kafka output, and one or more JSON Lines over TCP outputs.

You can optionally filter which tables (attribute groups) and which fields (attributes) to publish. Each output can specify a different filter.

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.

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

You can run OMEGAMON Data Connect on or off z/OS.

Of these components, only collection tasks and their configuration member, RKANPARU(KAYOPEN), are in the OMEGAMON runtime environment (RTE).

The load modules for the collection task, like other RTE load modules, are managed by whichever method you choose to use: PARMGEN or Monitoring Configuration Manager. However, *you* are responsible for managing all other components: OMEGAMON Data Broker, OMEGAMON Data Connect, and the three configuration members, including RKANPARU(KAYOPEN).

Related concepts

Overview of configurable parts
Before you start configuring OMEGAMON Data Provider, it's useful to understand the parts that you need to configure and their places in the architecture.

Related reference

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

Configuration parameters
OMEGAMON Data Provider has three configuration points: collection tasks, OMEGAMON Data Broker, and OMEGAMON Data Connect. Each point has its own configuration member containing a set of configuration parameters.

Characteristics of JSON output from OMEGAMON Data Connect
If you need to work directly with the 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 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:

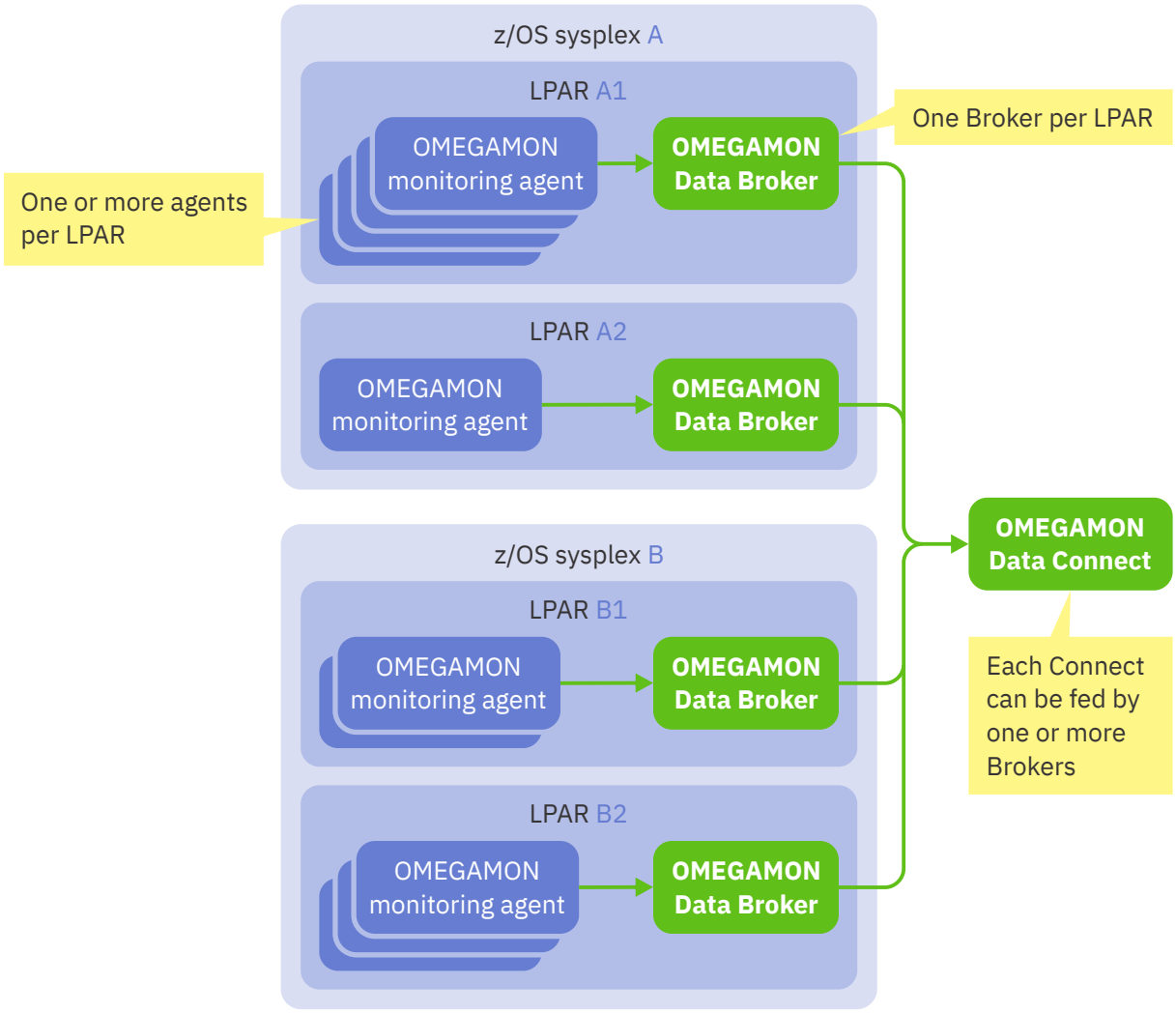

Figure 3. Example OMEGAMON Data Provider topology

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

You can configure each instance of OMEGAMON Data Broker to forward attributes 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:

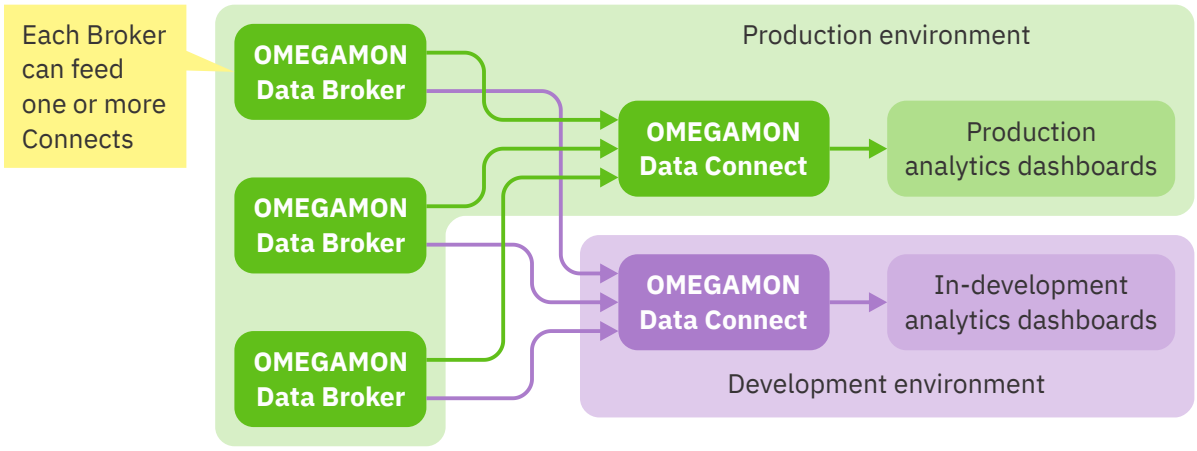

Figure 4. Example topology with separate instances of OMEGAMON Data Connect for production and development

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 attributes to production analytics dashboards.
- You don't have to drive a separate workload to send attributes to in-development analytics dashboards. The development environment receives attributes from the production workload.

Related reference

Configuration parameters

OMEGAMON Data Provider has three configuration points: collection tasks, OMEGAMON Data Broker, and OMEGAMON Data Connect. Each point has its own configuration member containing a set of configuration parameters.

OMEGAMON Data Broker configuration parameters

OMEGAMON Data Broker configuration parameters include the host name and port on which OMEGAMON Data Connect is listening.

Attribute destinations

OMEGAMON Data Provider introduces a choice of destination for attributes: the OMEGAMON persistent data store (PDS), OMEGAMON Data Provider, or both.

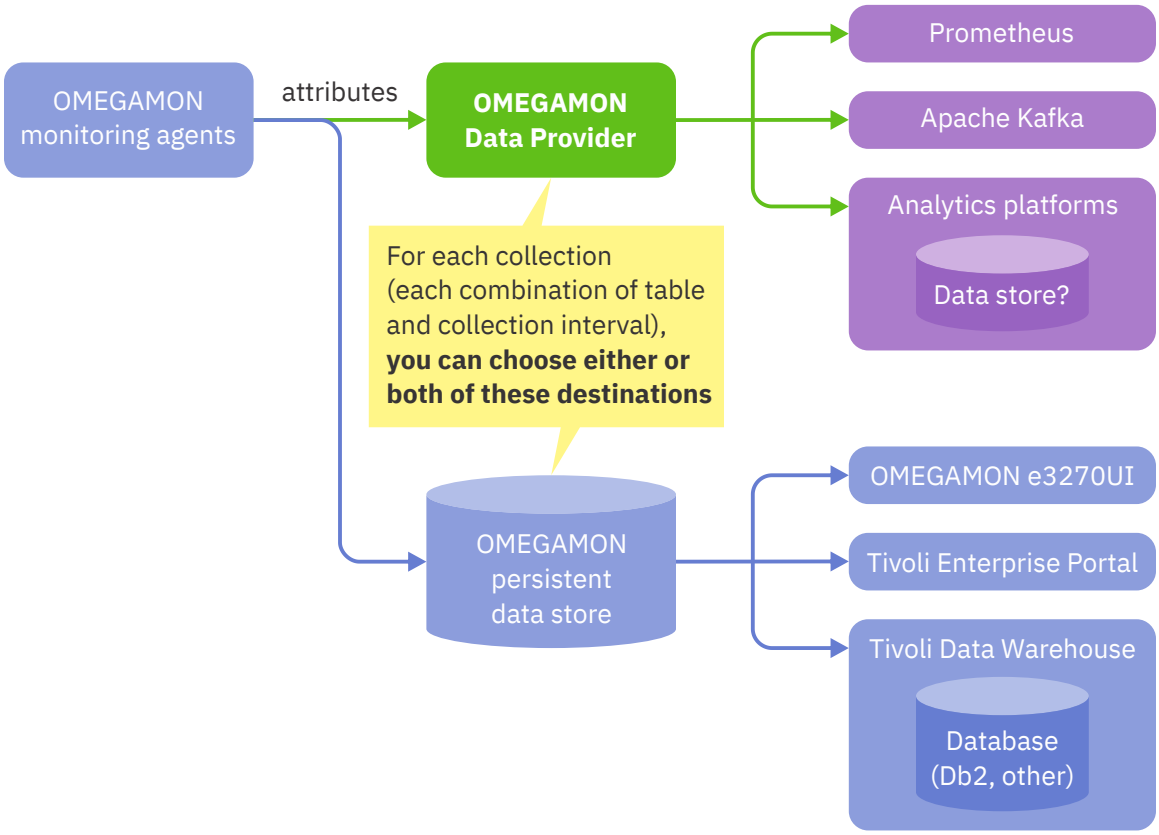

Figure 5. Choice of destinations for attributes

You can choose the destinations for each collection; more specifically, for each combination of attribute group (table) and collection interval. If you create multiple collections for the same table, but with different collection intervals, then you can choose different destinations for those collections.

If you want to view attributes in the OMEGAMON enhanced 3270 user interface (e3270UI) or the Tivoli Enterprise Portal (TEP) user interface, or store attributes in Tivoli Data Warehouse, then you must include the PDS as a destination.

To pass attributes directly through to OMEGAMON Data Provider without storing them on disk (in the PDS), specify OMEGAMON Data Provider as the only destination.

Related reference

OMEGAMON Data Provider collection configuration parameters

Collection tasks use OMEGAMON Data Provider collection configuration parameters to select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

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 applications or analytics platforms.

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

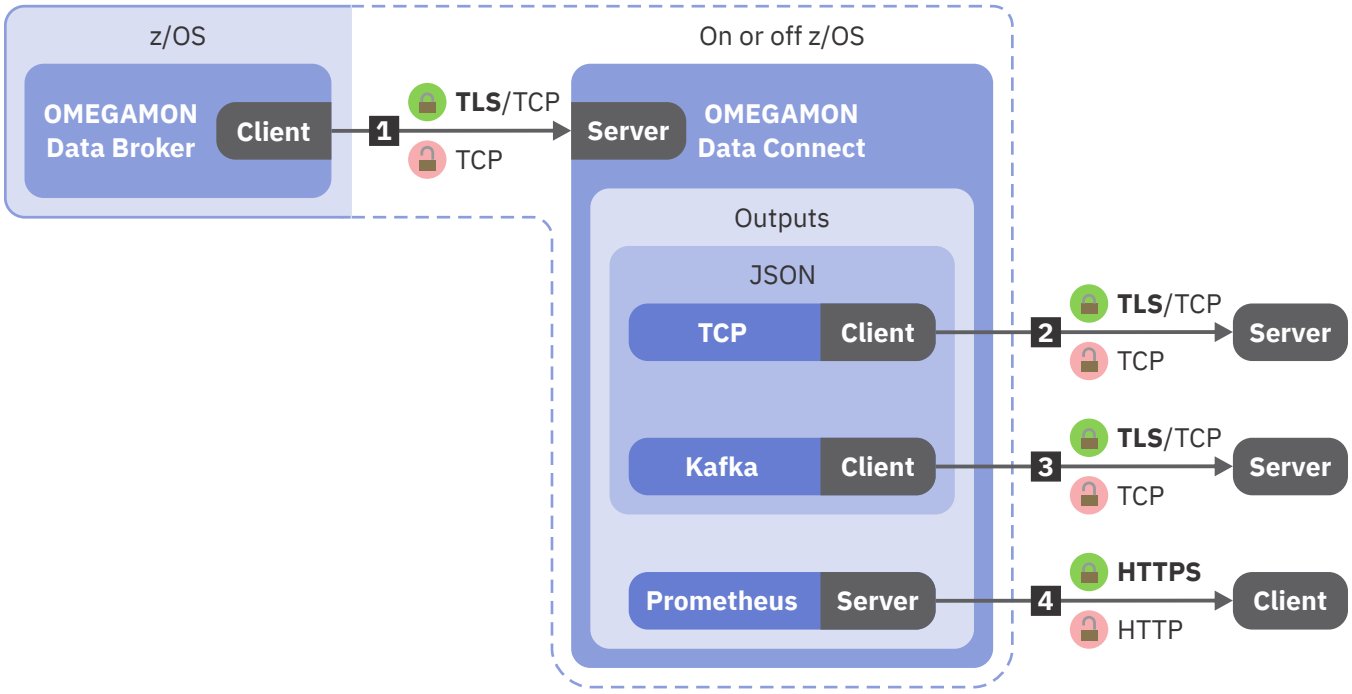

Figure 6. OMEGAMON Data Provider communication protocols with or without TLS

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 Kafka output	Server: Kafka server	TLS over TCP
4	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®.

Starter dashboards

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

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 attributes as JSON Lines over TCP to Logstash. You can configure Logstash to listen on a TCP port for that JSON Lines and forward the attributes to Elasticsearch.

Prerequisites for OMEGAMON Data Provider

Before installing OMEGAMON Data Provider, check that you have the prerequisite software.

Prerequisite OMEGAMON software

You must have an OMEGAMON product suite installed on z/OS that includes the following components:

- OMNIMON Base, minimum version 7.5.0, APAR/PTF level OA62052/UJ06872.
- IBM Z OMEGAMON Integration Monitor, minimum version 5.6.0.

  OMEGAMON Data Provider is a part of IBM Z OMEGAMON Integration Monitor.

  OMEGAMON Data Provider is packaged in its own FMID, HKOA110. You must have this FMID installed.
- One or more monitoring agents supported by OMEGAMON Data Provider.

To meet that requirement, you need one of the following product suites:

- IBM Z Monitoring Suite, minimum version 1.2.1
- IBM Z Service Management Suite, minimum version 2.1.1

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 Tivoli Enterprise Portal (TEP).

Java

OMEGAMON Data Connect is a Java application that requires Java 8, or later, 64-bit edition.

If you plan to run OMEGAMON Data Connect on z/OS: the supplied sample JCL to run OMEGAMON Data Connect assumes that you have the Java Batch Launcher (JZOS) installed. For details, see the JZOS documentation; for example, in the IBM SDK, Java Technology Edition 8 documentation.

Starter Elastic Kibana dashboards

You can get starter Elastic Kibana dashboards that visualize attributes from OMEGAMON Data Provider.

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-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 configure your analytics platform to ingest attributes from OMEGAMON Data Provider.
- You can develop your own application to process attributes from OMEGAMON Data Provider.

Zowe is not a prerequisite

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

Instead, the Zowe cross-memory server is supplied with OMEGAMON Data Provider in a single load module that has no dependencies on other Zowe components. OMEGAMON Data Provider has no Zowe requirements beyond the Zowe cross-memory server.

If you already have Zowe installed (minimum version: 1.24), then you can configure your existing Zowe cross-memory server to run OMEGAMON Data Broker. For details, see “Configuring OMEGAMON Data Broker” on page 21.

Related concepts

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

Related reference

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

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. However, you might want to install some components in other locations.

Read "OMEGAMON Data Provider architecture" on page 6 so that you understand the components involved.

Ensure that you have the prerequisite software, such as an OMEGAMON runtime environment (RTE) that you want to use with OMEGAMON Data Provider. See "Prerequisites for OMEGAMON Data Provider" on page 12.

You need to know the SMP/E target locations of OMEGAMON. In particular:

- The high-level qualifiers of the TKAN* MVS™ libraries, such as TKANMODP
- The z/OS UNIX path specified by the SMP/E **DDDEF** TKAYHFS

If you do not know these locations, then contact the person who installed OMEGAMON.

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

1. Ensure that your RTE load modules are at the prerequisite software levels.

   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 host OMEGAMON Data Broker in an existing instance of the Zowe cross-memory server on your system or create a new instance.

   If you already have Zowe installed (minimum version: 1.24), then you can configure your existing Zowe cross-memory server to run OMEGAMON Data Broker. For details, see "Configuring OMEGAMON Data Broker" on page 21.
3. Decide where you want to store the load modules for OMEGAMON Data Broker.

   OMEGAMON Data Broker involves the following load modules:

   ZWESIS01
   Zowe cross-memory server.

   KAYB0001
   OMEGAMON Data Broker, a plugin for the Zowe cross-memory server.

   KAYBNETL
   A load module used by OMEGAMON Data Broker.

   The **STEPLIB** of the Zowe cross-memory server must include the KAYB0001 and KAYBNETL load modules.

   If you decide to use an existing instance of the Zowe cross-memory server (minimum Zowe version: 1.24), then you must make the OMEGAMON Data Broker plugin available to the server. Copy the load modules TKANMODP(KAYB0001) and TKANMODP(KAYBNETL) to a data set in the **STEPLIB** of the job step that runs the Zowe cross-memory server. For example, the same library that contains the Zowe cross-memory server load module, ZWESIS01.

   Otherwise, use the Zowe cross-memory server load module supplied with OMEGAMON Data Provider:

   a. Rename the load module supplied with OMEGAMON Data Provider in TKANMODP(KAYSIS01) to the member name ZWESIS01.

      You *must* rename this module for the following reasons:

      The module loads itself into the link pack area (LPA) and relies on the ZWESIS01 module name to do that.

The current implementation does not support the use of an alias.

b. If you want to run OMEGAMON Data Broker from a different library than TKANMODP, then copy ZWESIS01, KAYB0001, and KAYBNETL to a different APF-authorized library.

4. Decide whether you want to run OMEGAMON Data Connect on or off z/OS.

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

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

   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 prerequisite OMEGAMON software was installed.

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

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

   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 `kay-110` directory to the z/OS UNIX path of your choice.

   If you want to run OMEGAMON Data Connect off z/OS, then transfer the binary file KAY11ZIP from z/OS UNIX to another platform, such as Linux®. Consider renaming the transferred copy with a `.zip` file extension. To create the OMEGAMON Data Connect installation directory `kay-110`, expand the compressed file.

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.

Overview of configurable parts

Before you start configuring OMEGAMON Data Provider, it's useful to understand the parts that you need to configure and their places in the architecture.

All OMEGAMON Data Provider configurable parts are text members that you can edit in a text editor such as the z/OS ISPF editor.

For each configurable part, OMEGAMON Data Provider supplies a sample member that you can use as a starting point.

OMEGAMON Data Provider involves two types of configurable parts:

- Members that *run* components:

  PROCLIB(ZWESIS01)
  : JCL procedure that runs the Zowe cross-memory server that hosts OMEGAMON Data Broker.

  PROCLIB(KAYCONN)
  : JCL procedure that runs OMEGAMON Data Connect.

  bin/connect
  : UNIX shell script that runs OMEGAMON Data Connect. You can use this either on or off z/OS.

- Members that *configure* components:

  RKANPARU(KAYOPEN)
  : A YAML document that specifies collection configuration parameters, such as which attribute groups to send to OMEGAMON Data Broker.

  PARMLIB(ZWESIP*xx*)
  : A plain-text member that specifies OMEGAMON Data Broker parameters, such as the host and port on which OMEGAMON Data Connect is listening.

  connect.yaml
  : A YAML document that specifies OMEGAMON Data Connect configuration parameters, such as the output method for attributes.

The following figure shows the configurable parts and their corresponding sample members:

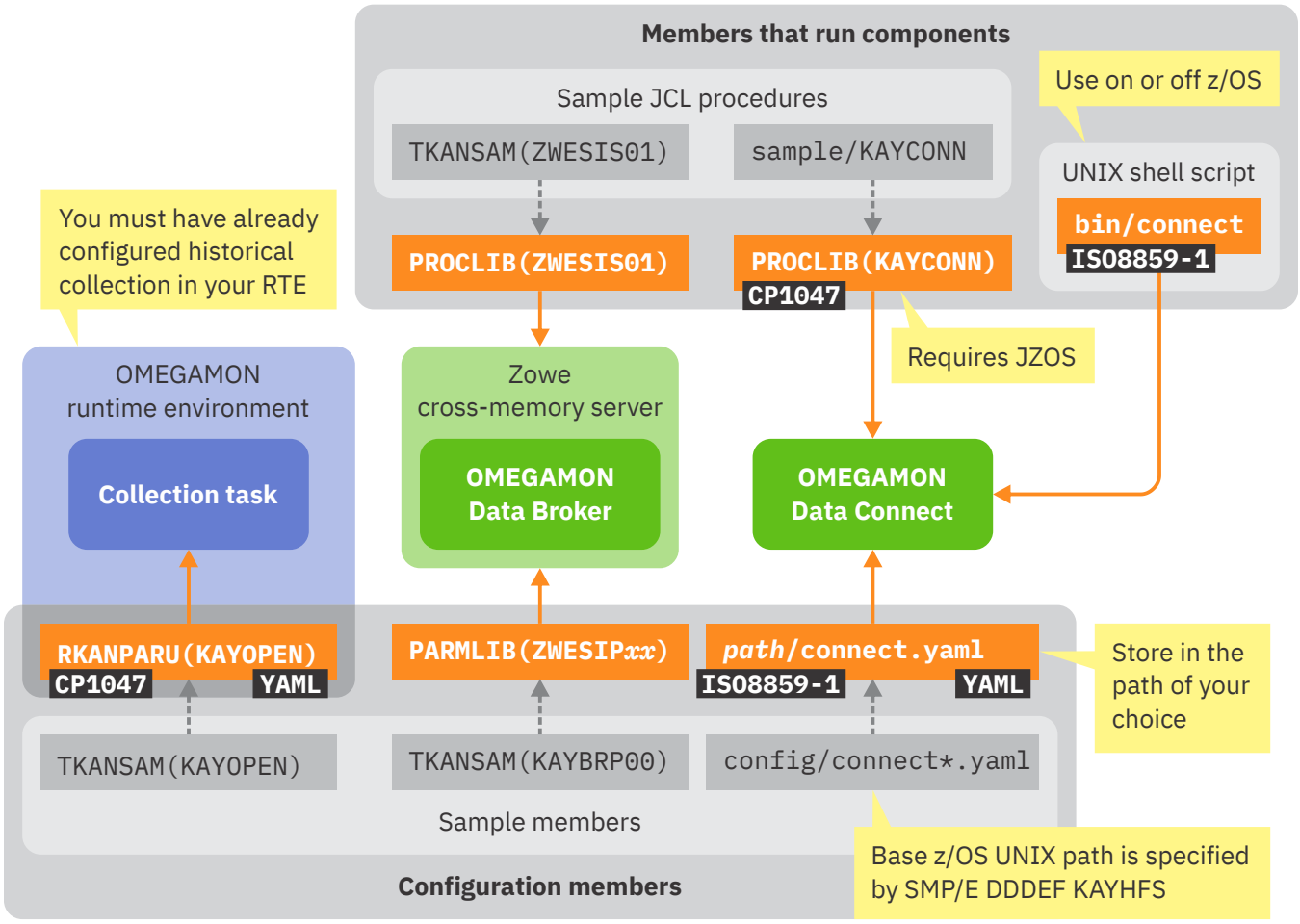

Figure 7. OMEGAMON Data Provider configurable parts

In the previous figure, annotations indicate the data format or character encoding of a member:

YAML
The member is a YAML document.

For details on the YAML format, including character encoding issues, see the topic for each document: RKANPARU(KAYOPEN) and connect.yaml.

CP1047
The member is encoded using EBCDIC code page 1047.

The JCL procedure that runs OMEGAMON Data Connect, PROCLIB(KAYCONN), contains an in-stream STDENV data set that must be encoded in CP1047.

ISO8859-1
The member is encoded using ISO8859-1 (CCSID 819), an 8-bit superset of ASCII.

Related concepts

OMEGAMON Data Provider architecture
OMEGAMON Data Provider extends OMEGAMON collection tasks and introduces two components: OMEGAMON Data Broker and OMEGAMON Data Connect.

Getting started with OMEGAMON Data Provider

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

These procedures configure OMEGAMON Data Provider to send attributes in JSON Lines format over TCP to a listening application or analytics platform, such as the Elastic Stack.

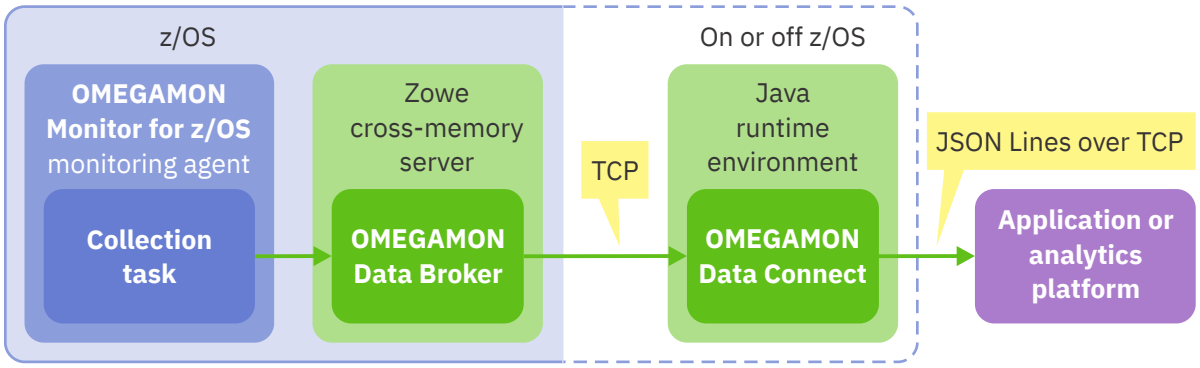

Figure 8. Architecture of the "getting started" exercise: sending JSON Lines over TCP

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

Use these procedures as an introductory exercise for configuring OMEGAMON Data Provider for other attribute groups and analytics platforms.

For comprehensive details on configuring each component of OMEGAMON Data Provider, see “Configuration parameters” on page 39.

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

To use OMEGAMON Data Provider to send attributes 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. For details, see “OMEGAMON Data Provider architecture” on page 6.

Configuring which collections to send to OMEGAMON Data Broker

You need to specify which historical collections to send to OMEGAMON Data Broker.

Historical collections are a prerequisite for using OMEGAMON Data Provider.

To create historical collections, use the OMEGAMON enhanced 3270 user interface (e3270UI) or 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 25. Otherwise, see the documentation for your analytics platform.

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

OMEGAMON Data Provider uses RKANPARU library member 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.

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: Do not maintain your primary copy of the KAYOPEN member in the RKANPARU library. Maintain it in a different location of your choice, and copy it from that location into RKANPARU. PARMGEN and Monitoring Configuration Manager do not manage the KAYOPEN member. Some actions of PARMGEN and Monitoring Configuration Manager, such as the **GENERATE** action of Monitoring Configuration Manager, delete existing members of the RKANPARU library of an RTE. Each time you perform such an action, you must copy the KAYOPEN member into the RKANPARU library.
2. Edit the KAYOPEN member in the permanent location you have chosen to maintain it.

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

Figure 9. Excerpt of example OMEGAMON Data Provider collection configuration member, KAYOPEN

1

In the broker.name key (the name child key of the broker key), specify the name of the Zowe cross-memory server that runs OMEGAMON Data Broker.

This name is the value of the **NAME** runtime parameter of the JCL **EXEC** statement that runs the Zowe cross-memory server program, ZWESIS01.

The default value is ZWESIS_STD.

2

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

Each entry must specify the following details:

- The k*pp* product code of the monitoring agent that owns the table.
- The table name.
- The collection interval, in minutes; or 0 to select all collections for this table, regardless of collection interval.
- Destinations. To send attributes to OMEGAMON Data Broker, the destinations must include the value open.

3. Copy the customized KAYOPEN 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 tasks

Reloading 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 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

OMEGAMON Data Provider collection configuration parameters
Collection tasks use OMEGAMON Data Provider collection configuration parameters to select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

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 plugin for the Zowe cross-memory server. You need to add OMEGAMON Data Broker parameters to the Zowe cross-memory server PARMLIB member. If you plan to host OMEGAMON Data Broker in a new instance of the Zowe cross-memory server, then you need to configure the JCL procedure that runs the server.

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

If you plan to use an existing instance of the Zowe cross-memory server to host OMEGAMON Data Broker, then you need to know the location of the JCL procedure that runs that server.

This task involves the following members supplied by the prerequisite OMEGAMON software:

TKANSAM(ZWESIS01)
JCL procedure that runs the Zowe cross-memory server. This member is required only if you are creating a new instance of the Zowe cross-memory server.

This member is a verbatim copy of a file in the public Zowe repository. This file does not refer to OMEGAMON Data Broker.

TKANSAM(KAYBRP00)
OMEGAMON Data Broker configuration parameters that you need to add to the Zowe cross-memory server configuration member, PARMLIB(ZWESIP*xx*).

1. If you have decided to host OMEGAMON Data Broker in an existing instance of the Zowe cross-memory server, append the contents of TKANSAM(KAYBRP00) to the existing PARMLIB(ZWESIP*xx*) member, and then skip to step "8" on page 23.

   Otherwise, if you have decided to use the Zowe cross-memory server load module provided with OMEGAMON Data Provider, proceed to the next step.
2. Modify the z/OS MVS program properties table (PPT) to make the Zowe cross-memory server run in key 4 and be non-swappable.
   a) Edit the PPT definition member SYS1.PARMLIB(SCHED*xx*).
      Add the following entry:

      ```
      PPT PGMNAME(ZWESIS01) KEY(4) NOSWAP
      ```

   b) Modify the PPT.
      Example MVS system command:

      ```
      SET SCH=xx
      ```

3. Ensure that the library that contains the Zowe cross-memory server load module, ZWESIS01, and the load modules for OMEGAMON Data Broker, KAYB0001 and KAYBNETL, is APF-authorized.

   To check the APF-authorization status of the library, enter the following MVS system command:

   ```
   D PROG,APF,DSNAME=loadlib
   ```

   where *loadlib* is the data set name of the library. For example, the TKANMODP library.

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

   ```
   SETPROG APF,ADD,DSNAME=loadlib,SMS
   ```

4. Copy the TKANSAM(KAYBRP00) member, renamed to member name ZWESIP00, to your choice of PARMLIB library. For example, SYS1.PARMLIB.
5. Copy the TKANSAM(ZWESIS01) JCL procedure to your choice of PROCLIB library. For example, SYS1.PROCLIB.
6. Edit the new copy of the PROCLIB(ZWESIS01) JCL procedure.

```
//ZWESIS01  PROC NAME='ZWESIS_STD',MEM=00,RGN=0M     1  2  3
…
//ZWESIS01 EXEC PGM=ZWESIS01,REGION=&RGN,            4
//         PARM='NAME=&NAME,MEM=&MEM'
//STEPLIB  DD   DSNAME=ZWES.SISLOAD,DISP=SHR         5
//PARMLIB  DD   DSNAME=ZWES.SISSAMP,DISP=SHR         6
//SYSPRINT DD   SYSOUT=*
```

Figure 10. JCL procedure that starts the Zowe cross-memory server, PROCLIB(ZWESIS01)

1

If you are only using this instance of the Zowe cross-memory server to run the OMEGAMON Data Broker plugin, then consider renaming the procedure with a prefix, such as **OMEG**, that matches related jobs; so that the Zowe cross-memory server job is included when you list job names with that prefix. For example, OMEGKAYB, where KAY is the component prefix for OMEGAMON Data Provider, and B is for "Broker".

2

The default name of the Zowe cross-memory server is ZWESIS_STD. You only need to change this value if you run more than one instance of the server on the same instance of z/OS.

3

The MEM parameter specifies the last two characters of the Zowe cross-memory server PARMLIB member name, ZWESIP*xx*. The default suffix is 00. You only need to change this value if you have different versions of this member in the PARMLIB.

4

The load module member name *must* be ZWESIS01. *Do not* refer to the original member name KAYSIS01 supplied with OMEGAMON Data Provider.

5

Edit the **STEPLIB DD** statement to refer to the location of the Zowe cross-memory server load module, ZWESIS01, the load modules for OMEGAMON Data Broker, KAYB0001 and KAYBNETL.

Note:

- The **STEPLIB** data set type must be partitioned data set extended (PDSE), not PDS.
- The **STEPLIB** data set, or data sets (if you decide to keep the ZWESIS01, KAYB0001, and KAYBNETL load modules in different data sets), 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).

6 Either edit the **PARMLIB DD** statement to refer to the PARMLIB library containing the ZWESIP*xx* member or, if you copied that member to the system PROCLIB, remove this statement.

7. Define the Zowe cross-memory server JCL procedure PROCLIB(ZWESIS01) as a started task.

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

   Example RACF commands:

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

8. Edit the OMEGAMON Data Broker parameters in the PARMLIB(ZWESIP*xx*) member.

   Set the values of the following parameters:

   KAY.CIDB.FWD.*forwarder_id*.SINK_HOST=*connect_host_name*
   Host name 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.

9. If you are using an existing instance of the Zowe cross-memory server, don't restart it yet with the updated configuration.

   We'll restart it later, after you have configured all components.

When the Zowe cross-memory server starts, it will load the OMEGAMON Data Broker plugin, and OMEGAMON Data Broker will connect to OMEGAMON Data Connect.

Related reference

OMEGAMON Data Broker configuration parameters
OMEGAMON Data Broker configuration parameters include the host name and port on which OMEGAMON Data Connect is listening.

Configuring OMEGAMON Data Connect

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

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: 64-bit, Java 8 or later.

1. Configure a z/OS started task, or a script off z/OS, to run OMEGAMON Data Connect.
   - On z/OS:
     a. Copy the JCL procedure `sample/KAYCONN` from the OMEGAMON Data Connect installation directory to your choice of MVS PROCLIB library. For example, SYS1.PROCLIB.
     b. Define the new PROCLIB(KAYCONN) procedure as a started task.

Example RACF commands:

```
RDEFINE STARTED KAYCONN.* STDATA(USER(OMEGSTC) GROUP(STCGROUP))
SETROPTS RACLIST(STARTED) REFRESH
```

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

- Off z/OS: Copy the sample Unix shell script `bin/connect` the location of your choice, as a starting point for your own site-specific script to run OMEGAMON Data Connect.

2. Edit your copy of the sample procedure or script to refer to paths on your system.

   Java home directory
   Specified by the `JAVAHOME` environment variable

   OMEGAMON Data Connect installation directory
   Specified by the `KAYHOME` environment variable

   OMEGAMON Data Connect configuration file
   Specified by the `spring.config.additional-location` option.

   Tip: To avoid service updates overwriting your edited version of this file, consider setting `spring.config.additional-location` in the OMEGAMON Data Connect startup procedure or script to a file path outside of the OMEGAMON Data Connect installation directory.

   Example (note the equal sign between the option name and value):

   ```
   --spring.config.additional-location=/u/kay/config/connect.yaml
   ```

3. Copy the supplied sample OMEGAMON Data Connect configuration file to the path that you specified in the `spring.config.additional-location` option, and then edit the parameters to match your site-specific values.

   Sample file path: `config/connect.yaml` in the OMEGAMON Data Connect installation directory.

   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

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

If you run OMEGAMON Data Connect on the same z/OS instance as OMEGAMON Data Broker, then you can specify `localhost` as the host name.

This value must match the OMEGAMON Data Broker parameter `KAY.CIDB.FWD.OM.SINK_HOST`.

2

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

This value must match the OMEGAMON Data Broker parameter `KAY.CIDB.FWD.OM.SINK_PORT`.

3. Destination host name or IP address to send data. For example, the host running Elastic Logstash.
4. Port on which the destination host is listening for JSON Lines over TCP.

Some analytics platforms might require, or provide specific support for, particular attributes. You might choose to filter the output from OMEGAMON Data Connect to send only those attributes. For information about some analytics platforms, see "Integrating analytics platforms with OMEGAMON Data Provider" on page 25. Otherwise, see the documentation for your analytics platform.

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 attributes 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 can configure the OMEGAMON Data Connect component of OMEGAMON Data Provider to send attributes as JSON Lines over TCP. You can also configure OMEGAMON Data Provider to send just the attributes that Instana specifically supports.

This documentation provides details for configuring OMEGAMON Data Provider to send attributes to Instana; specifically, to IBM Observability by Instana Application Performance Monitoring on z/OS.

For details of the Instana architecture and information about configuring Instana to ingest attributes from OMEGAMON Data Provider, see the IBM Observability by Instana Application Performance Monitoring on z/OS documentation.

Note: The information presented here about Instana is current as of 14 June 2022.

Overview of Instana integration with OMEGAMON Data Provider

Instana is one of several analytics platforms that can ingest attributes from OMEGAMON Data Provider as JSON Lines over TCP.

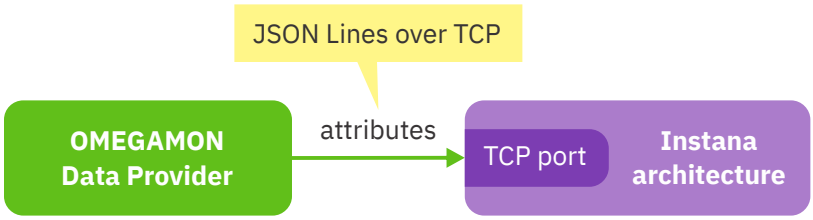

Figure 11. Instana ingests attributes from OMEGAMON Data Provider as JSON Lines over TCP

When sending to Instana, it's useful to send just the attributes that Instana specifically supports. If you send attributes that Instana does not specifically support, then Instana will ingest them, but not use them. If you omit attributes that Instana specifically supports, then you won't be taking full advantage of that support.

Historical data collections

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

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

Instana supports attribute groups from the following OMEGAMON monitoring agents:

IBM Z OMEGAMON Monitor for z/OS (product code: km5)

Table 2. Historical data collections for Instana: z/OS

table_name field value	**Attribute group**	**Collection interval (minutes)**
m5stgcdth	Common Storage Utilization History	5
m5stgfdth	Real Storage Utilization History	5
syscpuutil	System CPU Utilization	1

IBM Z OMEGAMON for CICS (kc5)

Table 3. Historical data collections for Instana: CICS

table_name field value	**Attribute group**	**Collection interval (minutes)**
cicsrov	CICSplex Overview	1

IBM OMEGAMON for Db2 Performance Expert on z/OS (kd5)

Table 4. Historical data collections for Instana: Db2

table_name field value	**Attribute group**	**Collection interval (minutes)**
dp_sy_exc	Db2 System States	1
opersys	z/OS System Statistics	1
db2lkconf	Local Db2 Lock Conflict	1
dp_srm_sub	Db2 SRM Subsystem	1
dp_ci_excs	Db2 CICS Exceptions	1
dp_im_conn	Db2 IMS Connections	1

OMEGAMON Data Provider configuration members

The following two configuration members specify which 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.

connect.yaml
Specifies which attribute groups, or specific attributes, to send to each output of OMEGAMON Data Connect, such as Instana.

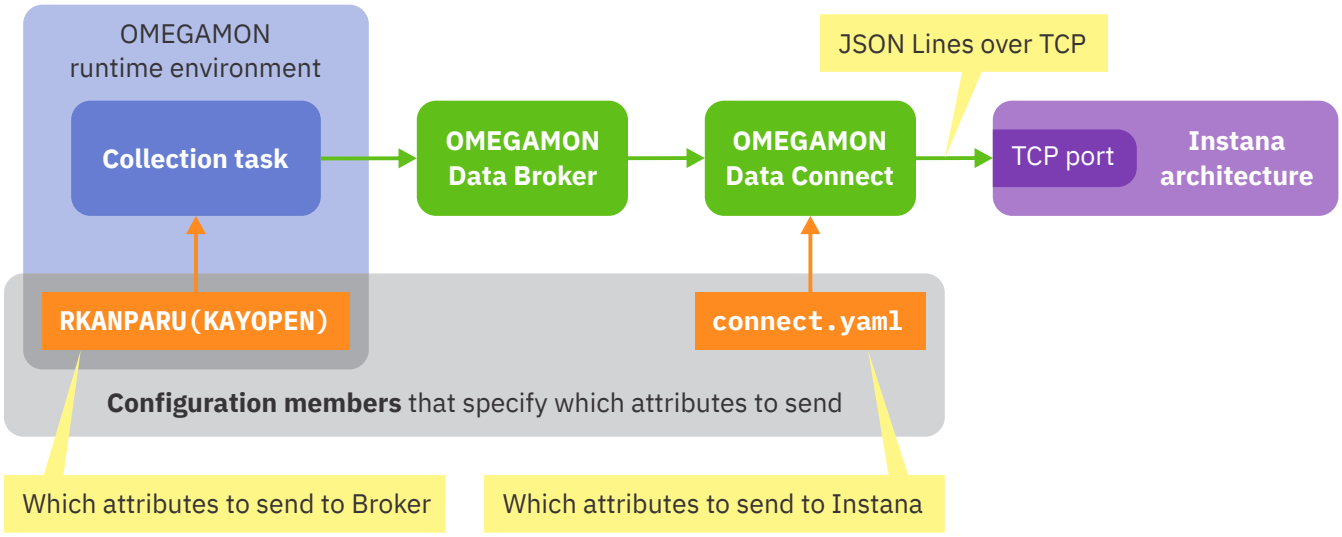

Figure 12. Configuring which attributes OMEGAMON Data Provider sends, and to where

Example KAYOPEN member

The "getting started" task “Configuring which collections to send to OMEGAMON Data Broker” on page 19 describes editing the KAYOPEN member. When you reach that step, use the following example KAYOPEN member, which matches the historical collections listed in the previous tables.

```
broker:
  name: ZWESIS_STD
collections:
  - product: km5
    table: syscpuutil
    interval: 0
    destination:
      - pds
      - open
  - product: km5
    table: m5stgcdth
    interval: 0
    destination:
      - pds
      - open
  - product: km5
    table: m5stgfdth
    interval: 0
    destination:
      - pds
      - open
  - product: kc5
    table: cicsrov
    interval: 0
    destination:
      - pds
      - open
  - product: kd5
    table: dp_sy_exc
    interval: 0
    destination:
      - pds
      - open
  - product: kd5
    table: opersys
    interval: 0
    destination:
```

```
      - pds
      - open
  - product: kd5
    table: db2lkconf
    interval: 0
    destination:
      - pds
      - open
  - product: kd5
    table: dp_srm_sub
    interval: 0
    destination:
      - pds
      - open
  - product: kd5
    table: dp_ci_excs
    interval: 0
    destination:
      - pds
      - open
  - product: kd5
    table: dp_im_conn
    interval: 0
    destination:
      - pds
      - open
```

Example `connect.yaml`

The "getting started" task “Configuring OMEGAMON Data Connect” on page 23 describes editing `connect.yaml`. When you reach that step, specify a TCP output that refers to the host and port on which Instana is listening, and refer to the Instana filter include file that is embedded in OMEGAMON Data Connect. For example:

```
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
          filter: # Refers to a file embedded in the Connect JAR file
            enabled: true
            include: filters/instana.yaml
```

Embedded `instana.yaml` filter include file

You don't *need* to know the contents of the Instana filter include file that is embedded in OMEGAMON Data Connect. You just refer to the file path, as shown in the previous `connect.yaml` listing.

However, for interest, here is the contents of that file as supplied with OMEGAMON Data Provider, APAR level OA63141. Note that the filter corresponds to the collections listed previously in the example KAYOPEN member.

```
enabled: true
products:
  km5:
    tables:
      syscpuutil:
        enabled: true
```

```
      m5stgcdth:
        enabled: true
      m5stgfdth:
        enabled: true
  kc5:
    tables:
      cicsrov:
        enabled: true
  kd5:
    tables:
      dp_sy_exc:
        enabled: true
      opersys:
        enabled: true
      db2lkconf:
        enabled: true
      dp_srm_sub:
        enabled: true
      dp_ci_excs:
        enabled: true
      dp_im_conn:
        enabled: true
```

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 attributes as JSON Lines over TCP to Logstash. You can configure Logstash to listen on a TCP port for that JSON Lines and forward the attributes to Elasticsearch.

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 Data Provider.

Related reference

TCP output parameters
OMEGAMON Data Connect TCP output parameters specify one or more destinations ("sinks") for sending attributes 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 13. 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 14. 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.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.

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 attributes 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 attributes 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 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 value of the JSON key `table_name`.

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

Each line of JSON Lines from OMEGAMON Data Connect contains a `table_name` field that identifies the OMEGAMON attribute table too which the data belongs. You can use this field to set the Splunk source type.

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 to value of table_name field
REGEX = \"table_name\":\"([^\"]+)\"
FORMAT = sourcetype::omegamon_$1
DEST_KEY = MetaData:Sourcetype
```

Starting OMEGAMON Data Provider

Starting OMEGAMON Data Provider involves starting the related components: OMEGAMON Data Connect, OMEGAMON Data Broker, and the OMEGAMON runtime environment that collects attributes.

You should have already configured and started an analytics platform, such as the Elastic Stack, or an application or tool to listen for JSON Lines over TCP from OMEGAMON Data Connect. You should have tested that software, and confirmed that it successfully receives JSON Lines on that TCP port. That software should be actively listening now.

You should have already tested that your OMEGAMON runtime environment collects attributes in the persistent data store without OMEGAMON Data Provider. This confirms that you have successfully configured historical data collection; this is a prerequisite for using OMEGAMON Data Provider.

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 OMEGAMON components in order from "downstream" to "upstream":

1. OMEGAMON Data Connect
2. OMEGAMON Data Broker
3. 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.

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 the OMEGAMON Data Connect Java application.
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 host:port
   ```

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 ZWESIS01,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 Store 'OMEGAMON' 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 OMEGAMON runtime environment, if it is not already running.

   Use your site-specific procedures to start the runtime environment jobs.

6. 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*

   KAYC0008I indicates the first time, either since starting or since its configuration was refreshed by a **MODIFY** command, that this instance of OMEGAMON Data Connect has received data for this table.

   KAYC0033I indicates the first time, either since starting or since its configuration was refreshed by a **MODIFY** command, 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 attributes are 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

No KPQH037I or KPQH038I message for a table

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**.

Reloading 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.

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).

Enter the following **MODIFY** command for each job:

```
F job_name,KPQ,RELOAD_CONFIG,KAY
```

To confirm the configuration changes, read the KAYL0005I messages in the RKLVLOG output data set of the job.

Related tasks

Configuring which collections to send to OMEGAMON Data Broker
You need to specify which historical collections to send to OMEGAMON Data Broker.

Displaying OMEGAMON Data Broker status

You can enter MVS **MODIFY** system commands to display information about the status of OMEGAMON Data Broker.

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 ZWESIS01,D(KAYB) FWD
  ```

- To display the status of stores, such as how many records OMEGAMON Data Broker has sent to OMEGAMON Data Connect:

  ```
  F ZWESIS01,D(KAYB) STORE
  ```

where:

- ZWESIS01 is the name of the Zowe cross-memory server job.
- KAYB identifies the OMEGAMON Data Broker plugin as the target of the command.

Restarting OMEGAMON Data Connect

If you are running OMEGAMON Data Connect on z/OS, then you can enter an MVS **MODIFY** system command to restart it. Restarting OMEGAMON Data Connect reloads its configuration parameters.

Attention: Restarting OMEGAMON Data Connect flushes unsent records. Flushed records are lost, not sent to any destination.

Enter the following **MODIFY** command:

```
F OMEGCONN,APPL=RESTART
```

where OMEGCONN is the name of the OMEGAMON Data Connect job.

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 attributes to output.

Stopping components on z/OS

To stop an instance of OMEGAMON Data Broker, or an instance of OMEGAMON Data Connect that is running on z/OS, enter an MVS **STOP** system command.

There is no **MODIFY** command for this action; use the **STOP** command.

Enter the following **STOP** command:

```
P job_name
```

where *job_name* is the name of one of the following jobs:

- The Zowe cross-memory server job that is running OMEGAMON Data Broker
- OMEGAMON Data Connect

Adding more 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.

1. Edit the RKANPARU(KAY0PEN) member to select the additional collections.

   Tip: Edit KAY0PEN 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 the OMEGAMON Data Connect configuration and then restart OMEGAMON Data Connect.

   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, use the OMEGAMON enhanced 3270 user interface (e3270UI) or Tivoli Enterprise Portal (TEP). For more information about creating historical collections, see the OMEGAMON documentation for e3270UI and TEP.
5. Check the RKLVL0G output data set of the affected monitoring agent jobs for KPQH038I messages.
6. Check the STD0UT 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 which collections to send to OMEGAMON Data Broker
You need to specify which historical collections to send to OMEGAMON Data Broker.

Related information

No KPQH037I or KPQH038I message for a table

Configuration parameters

OMEGAMON Data Provider has three configuration points: collection tasks, OMEGAMON Data Broker, and OMEGAMON Data Connect. Each point has its own configuration member containing a set of configuration parameters.

The following figure shows the three configuration points of OMEGAMON Data Provider and their corresponding configuration members:

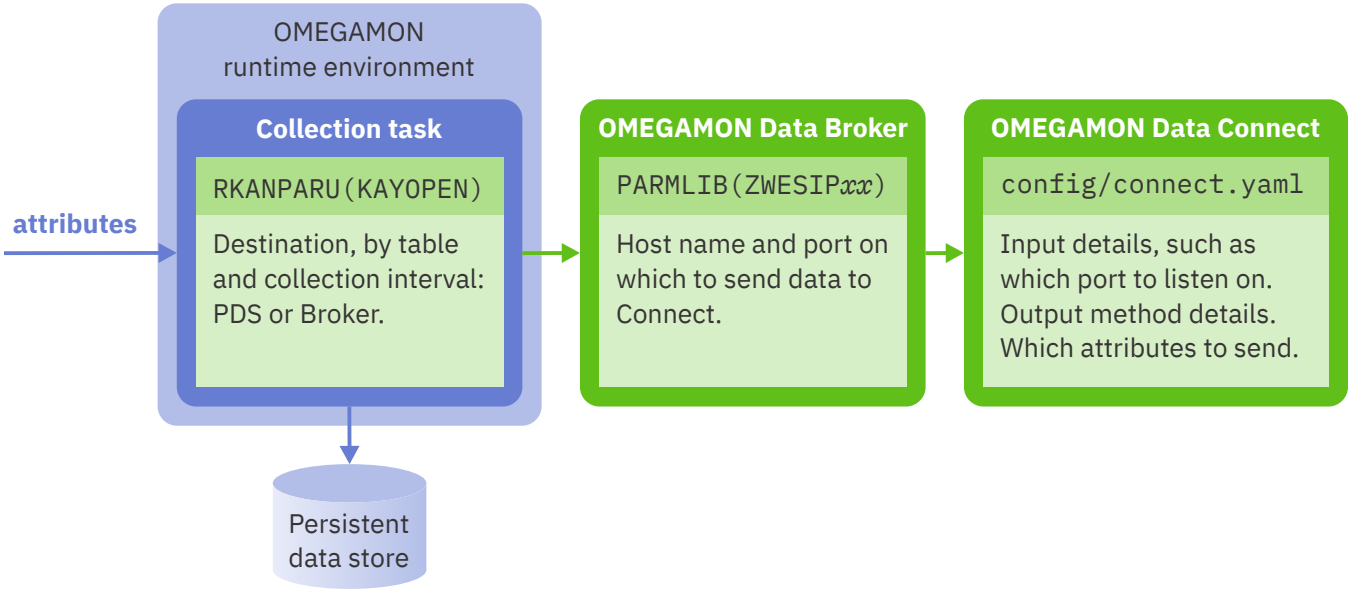

Figure 15. OMEGAMON Data Provider configuration points: Collection, Broker, Connect

Table 5. OMEGAMON Data Provider configuration points, configuration members, and sample members

Configuration point	Configuration member	Sample member
Collection task	RKANPARU(KAYOPEN) in your runtime environment	TKANSAM(KAYOPEN)
OMEGAMON Data Broker	PARMLIB(ZWESIP*xx*)	TKANSAM(KAYBRP00)
OMEGAMON Data Connect	config/connect.yaml in the OMEGAMON Data Connect installation directory	config/connect*.yaml

None of these configuration members is managed by PARMGEN or Monitoring Configuration Manager.

OMEGAMON Data Provider collection configuration parameters

Collection tasks use OMEGAMON Data Provider collection configuration parameters to select collections and set their destinations: the OMEGAMON persistent data store (PDS), OMEGAMON Data Broker, both, or none.

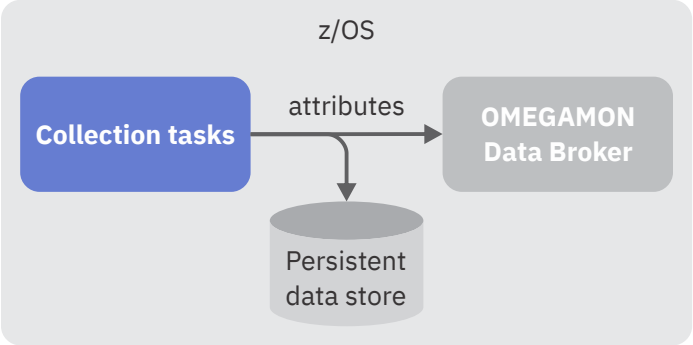

Figure 16. OMEGAMON Data Provider collection configuration parameters control where attributes are sent

Selecting versus creating collections

These parameters *select* collections; they do not *create* collections.

Historical collections are a prerequisite for using OMEGAMON Data Provider.

To create historical collections, use the OMEGAMON enhanced 3270 user interface (e3270UI) or 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: kpp # 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: Use a YAML validator to check that your configuration file conforms to YAML syntax.

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

This configuration member is optional. If you omit it, then OMEGAMON Data Provider is dormant and attributes from historical collections are sent to PDS only.

If you choose to specify a configuration member, then it must be member name KAYOPEN in the RKANPARU library of your OMEGAMON runtime environment (RTE).

Attention: Do not maintain your primary copy of the KAYOPEN member in the RKANPARU library. Maintain it in a different location of your choice, and copy it from that location into RKANPARU. PARMGEN and Monitoring Configuration Manager do not manage the KAYOPEN member. Some actions of PARMGEN and Monitoring Configuration Manager, such as the **GENERATE** action of Monitoring Configuration Manager, delete existing members of the RKANPARU library of an RTE. Each time you perform such an action, you must copy the KAYOPEN member into the RKANPARU library.

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 ZWESIS01 program (corresponding default procedure and job name: ZWESIS01).

Typical value: ZWESIS_STD

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 this choice of destinations, see “Attribute destinations” on page 10.

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. 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: ZWESIS_STD
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.

A similar example set of parameters is supplied in the KAYOPEN member of the TKANSAM sample library.

Example: Specific collection intervals

The following example only selects collections with the cited collection intervals.

```
broker:
  name: ZWESIS_STD
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: ZWESIS_STD
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: Combination of wildcard and specific collection intervals

The following example sends all collections for CICS (`kc5`) table `kcpplx` to the PDS; it also sends a collection for that table with a collection interval of 1 minute to OMEGAMON Data Broker.

```
broker:
  name: ZWESIS_STD
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
```

The following similar example sends all collections for table `kcpplx` to OMEGAMON Data Broker; it also sends a collection for that table with a collection interval of 1 minute to the PDS.

```
broker:
  name: ZWESIS_STD
collections:
  - product: kc5
    table: kcpplx
    interval: 0
    destination:
      - open # This line is the only difference from the previous example
  - product: kc5
    table: kcpplx
    interval: 1 # Specific value takes precedence over wildcard (0)
    destination:
      - pds
      - open
```

Related concepts

Attribute destinations

OMEGAMON Data Provider introduces a choice of destination for attributes: the OMEGAMON persistent data store (PDS), OMEGAMON Data Provider, or both.

Related tasks

Configuring which collections to send to OMEGAMON Data Broker

You need to specify which historical collections to send to OMEGAMON Data Broker.

Related reference

Monitoring agents supported by OMEGAMON Data Provider

OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Related information

No KPQH037I or KPQH038I message for a table

OMEGAMON Data Broker configuration parameters

OMEGAMON Data Broker configuration parameters include the host name and port on which OMEGAMON Data Connect is listening.

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*.

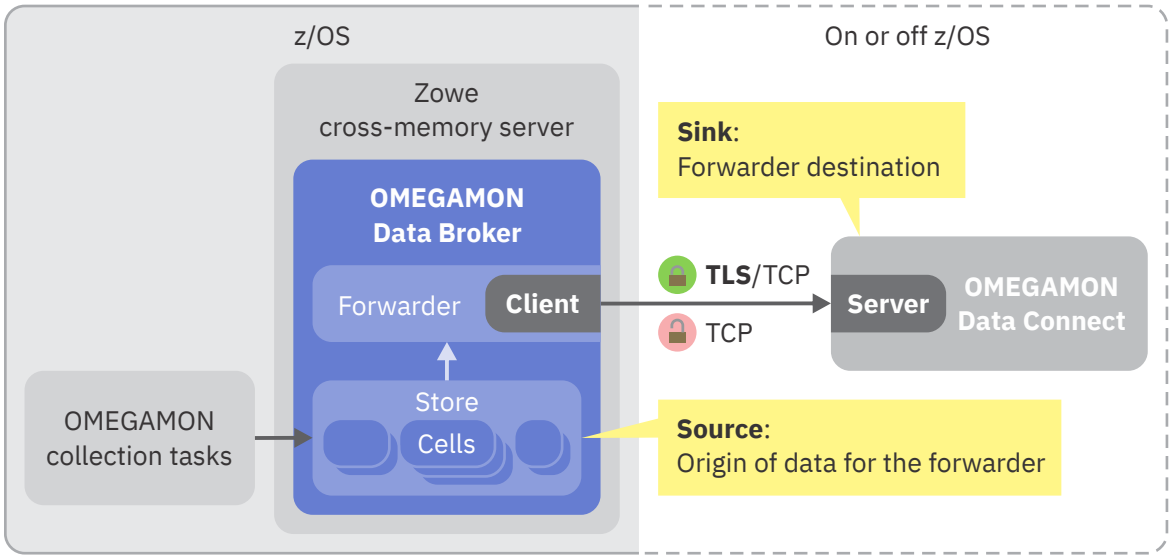

Figure 17. OMEGAMON Data Broker configuration points: store, forwarder, and output ("sink")

OMEGAMON Data Broker receives attributes from collection tasks into an internal store, and then forwards the attributes to OMEGAMON Data Connect.

To forward attributes to OMEGAMON Data Connect, you configure a *forwarder* with the store as its *source* and OMEGAMON Data Connect as its *sink*.

You can configure one or more forwarders. Each forwarder uses the same store as its source but sends data to a different sink; a different instance of OMEGAMON Data Connect:

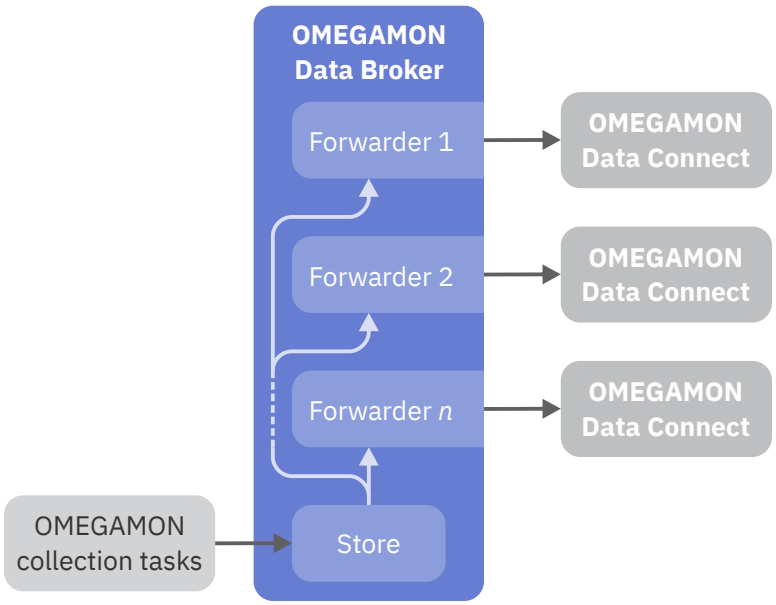

Figure 18. OMEGAMON Data Broker configuration: one store, one or more forwarders

Format

```
* General parameters
* Register the OMEGAMON Data Broker plugin (load module)
ZWES.PLUGIN.CIDB=KAYB0001
* Enable the forwarder subsystem
KAY.CIDB.FWD=ON

* Forwarder parameters
```

```
KAY.CIDB.FWD.forwarder_id.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.forwarder_id.SINK_HOST=connect_host_name
KAY.CIDB.FWD.forwarder_id.SINK_PORT=connect_port
* Optional timeout and retry parameters
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.CONNECT_RETRY_INTERVAL=seconds
KAY.CIDB.FWD.forwarder_id.MAX_CONNECT_RETRY_ATTEMPTS=number
* Other optional parameters
KAY.CIDB.FWD.forwarder_id.RECORD_QUEUE_LIMIT=records
KAY.CIDB.FWD.forwarder_id.LOGOPTS=--verbosity log_level
* 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...

* Store parameters
KAY.CIDB.STORE.store_id.NAME=OMEGAMON
* As a starting point, use the cell definitions in
* sample member TKANSAM(KAYBRP00)
KAY.CIDB.STORE.store_id.CELL.cell_id.SIZE=bytes
KAY.CIDB.STORE.store_id.CELL.cell_id.CAPACITY=number
* More OMEGAMON store cell definitions...
```

OMEGAMON Data Broker configuration parameter names are case-sensitive.

Location

OMEGAMON Data Broker configuration parameters are stored in the configuration member of the Zowe cross-memory server, PARMLIB(ZWESIP*xx*).

A configuration member named ZWESIP*xx* must exist in one of the following data sets:

- The data set specified by the PARMLIB ddname of the job step that runs the Zowe cross-memory server program, ZWESIS01.
- If that job step does not specify a PARMLIB ddname, the system PARMLIB. For example, SYS1.PARMLIB.

The last two characters of the configuration member name are determined by the optional **MEM** runtime parameter of the Zowe cross-memory server. The following example uses the configuration member ZWESIP02:

```
//ZWESIS01  PROC NAME='ZWESIS_STD',MEM=02,RGN=0M
```

The default value of **MEM** is 00. If you omit the **MEM** runtime parameter, the program uses configuration member ZWESIP00.

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.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 *forwarder_id*, *store_id*, or *cell_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 forwarder (FWD), store (STORE), or cell (CELL). 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:

```
OM
1
2
A
B
```

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 use the Zowe cross-memory server only to host OMEGAMON Data Broker, then the Zowe cross-memory server configuration member, PARMLIB(ZWESIP*xx*), can contain only the following parameters:

- ZWES.PLUGIN.CIDB=KAYB0001, to register the OMEGAMON Data Broker plugin.
- KAY.CIDB-namespace OMEGAMON Data Broker configuration parameters described here.

You do not need to specify any other ZWES-namespace parameters for the Zowe cross-memory server itself.

Splitting long parameter values over multiple lines

Some parameters in the KAY.CIDB.FWD namespace can have long values.

However, each record of the Zowe cross-memory server configuration member can contain a maximum of only 71 characters.

To split long values of KAY.CIDB.FWD-namespace parameters over multiple lines, use a backslash (\) as a line continuation character. Example:

```
KAY.CIDB.FWD.OM.KEYRING=\
  /u/my/long/directory/path/to/\
  a-long-file-name.p12
```

Leading spaces on continuation lines are ignored.

Use the sample configuration member

As a starting point, use sample configuration member TKANSAM(KAYBRP00).

For a connection without Transport Security Layer (TLS), you only need to change the values of two parameters in that sample member:

```
KAY.CIDB.FWD.OM.SINK_HOST
KAY.CIDB.FWD.OM.SINK_PORT
```

General parameters

The following general parameters are required:

ZWES.PLUGIN.CIDB=KAYB0001
Identifies the member name of the OMEGAMON Data Broker load module.

OMEGAMON Data Broker is a Zowe cross-memory server plugin.

The OMEGAMON Data Broker load module KAYB0001 must be a member of the data set specified by the STEPLIB ddname of the job step that runs the Zowe cross-memory server program, ZWESIS01.

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.

Forwarder parameters

You can define one or more forwarders. Use a different *forwarder_id* to group the parameters for each forwarder.

The following parameters are required:

KAY.CIDB.FWD.*forwarder_id*.SOURCE_STORE=OMEGAMON
The name of the OMEGAMON Data Broker store to which OMEGAMON collection tasks sends data.

For OMEGAMON Data Provider, you must specify the store name OMEGAMON.

KAY.CIDB.FWD.*forwarder_id*.SINK_HOST=*connect_host_name*
Host name 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.

The following parameters are optional:

KAY.CIDB.FWD.*forwarder_id*.CONNECT_TIMEOUT=*seconds*
Time in seconds to wait to establish a connection to OMEGAMON Data Connect. Default: 5.

KAY.CIDB.FWD.*forwarder_id*.RECEIVE_TIMEOUT=*seconds*
Receive timeout in seconds. Default: 5.

KAY.CIDB.FWD.*forwarder_id*.SEND_TIMEOUT=*seconds*
Send timeout in seconds. Default: 0 (indefinite).

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.

The following optional parameters are deliberately omitted from the sample member, because their default values are typically suitable:

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.

If the queue reaches this limit, then it stops expanding and becomes circular, fixed-length: each new record overwrites the oldest record. If the queue length later decreases, it reverts to adding instead of overwriting records.

If OMEGAMON Data Connect is unavailable, or is unable to receive data at the rate that OMEGAMON Data Broker receives data, then, potentially, older records in the queue might be lost; overwritten by new records before they can be forwarded.

KAY.CIDB.FWD.*forwarder_id*.LOGOPTS=--verbosity *log_level*
The logging level of activity for this forwarder. The default value is `0`.

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

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

Example:

```
KAY.CIDB.FWD.OM.LOGOPTS=--verbosity 7
```

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.

Forwarder SSL parameters

The following parameters are relevant only if you use TLS to secure the connection between OMEGAMON Data Broker and OMEGAMON Data Connect:

KAY.CIDB.FWD.*forwarder_id*.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 this parameter.

KAY.CIDB.FWD.*forwarder_id*.FIPS=ON|OFF
Sets z/OS System SSL Federal Information Processing Standards (FIPS) mode. Default: 0FF. 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.

KAY.CIDB.FWD.*forwarder_id*.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
```

KAY.CIDB.FWD.*forwarder_id*.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.

KAY.CIDB.FWD.*forwarder_id*.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.

KAY.CIDB.FWD.*forwarder_id*.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.

KAY.CIDB.FWD.*forwarder_id*.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.

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.

Store parameters

Typically, you do not need to understand store parameters in detail. Unless you have a specific reason to use different values, use the values supplied in sample member TKANSAM(KAYBRP00).

Tip: You only need to specify one set of store parameters, regardless of the number of forwarders.

OMEGAMON Data Broker places each incoming record into a cell in a store. Cells can be various sizes. Store parameters specify the different sizes of cell in the store and the initial number of cells of each size.

The following parameters are required:

KAY.CIDB.STORE.*store_id*.NAME=OMEGAMON
Defines a store named OMEGAMON.

For OMEGAMON Data Provider, you must specify the store name OMEGAMON.

KAY.CIDB.STORE.*store_id*.CELL.*cell_id*.SIZE=*bytes*
Cell size, in bytes.

The *cell_id* 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.

KAY.CIDB.STORE.*store_id*.QUEUE.CAPACITY=*number*
The initial number of cells that the store's queues can contain. Default: 10000.

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.

Example: Forwarding to OMEGAMON Data Connect without TLS

The following example configures OMEGAMON Data Broker to send attributes to OMEGAMON Data Connect that is running on the same z/OS instance as OMEGAMON Data Broker (localhost) and listening on port 15351:

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

KAY.CIDB.FWD.OM.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.OM.SINK_HOST=localhost
KAY.CIDB.FWD.OM.SINK_PORT=15351

KAY.CIDB.STORE.OM.NAME=OMEGAMON
KAY.CIDB.STORE.OM.CELL.1.SIZE=128
KAY.CIDB.STORE.OM.CELL.1.CAPACITY=1000
KAY.CIDB.STORE.OM.CELL.2.SIZE=256
KAY.CIDB.STORE.OM.CELL.2.CAPACITY=5000
KAY.CIDB.STORE.OM.CELL.3.SIZE=512
KAY.CIDB.STORE.OM.CELL.3.CAPACITY=5000
KAY.CIDB.STORE.OM.CELL.4.SIZE=1024
KAY.CIDB.STORE.OM.CELL.4.CAPACITY=1000
KAY.CIDB.STORE.OM.CELL.5.SIZE=2048
KAY.CIDB.STORE.OM.CELL.5.CAPACITY=1000
KAY.CIDB.STORE.OM.CELL.6.SIZE=4096
KAY.CIDB.STORE.OM.CELL.6.CAPACITY=1000
KAY.CIDB.STORE.OM.CELL.7.SIZE=8192
KAY.CIDB.STORE.OM.CELL.7.CAPACITY=200
```

In this example, the forwarder and the store have the same ID, OM. This common value has no significance; it does not define a relationship between the forwarder and the store. The relationship between the forwarder and the store is defined by the forwarder SOURCE_STORE and the store NAME parameters.

A similar example set of OMEGAMON Data Broker parameters is supplied in the sample member TKANSAM(KAYBRP00).

Some parameter values in the previous example listing, such as **CAPACITY** and **SIZE**, might differ from the values in the sample member. Use the values in the sample member.

Example: Forwarding to multiple instances of OMEGAMON Data Connect

To define additional forwarders, add the following parameters to the first example:

```
# Second instance of OMEGAMON Data Connect
KAY.CIDB.FWD.OM2.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.OM2.SINK_HOST=analytics2.example.com
KAY.CIDB.FWD.OM2.SINK_PORT=15351

# Third instance of OMEGAMON Data Connect
KAY.CIDB.FWD.OM3.SOURCE_STORE=OMEGAMON
KAY.CIDB.FWD.OM3.SINK_HOST=analytics3.example.com
KAY.CIDB.FWD.OM3.SINK_PORT=15351
```

Each forwarder has the same source store, but different FWD.*forwarder_id* and sink details.

Example: Forwarding to OMEGAMON Data Connect with TLS using a RACF key ring

Add the following parameters to the first example:

```
KAY.CIDB.FWD.OM.SECURITY=TLSv1.2
KAY.CIDB.FWD.OM.FIPS=ON
KAY.CIDB.FWD.OM.KEYRING=ZWESIS01/ZWESring
```

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 ZWESring, owned by user ZWESIS01 (the user that runs the Zowe cross-memory server instance that hosts the OMEGAMON Data Broker plugin).
- 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.OM.CERTLABEL=OMDPcert
```

where OMDPcert is the label (alias) of the client certificate in the key ring.

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 plugin for the Zowe cross-memory server. You need to add OMEGAMON Data Broker parameters to the Zowe cross-memory server PARMLIB member. If you plan to host

OMEGAMON Data Broker in a new instance of the Zowe cross-memory server, then you need to configure the JCL procedure that runs the server.

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 attributes to output.

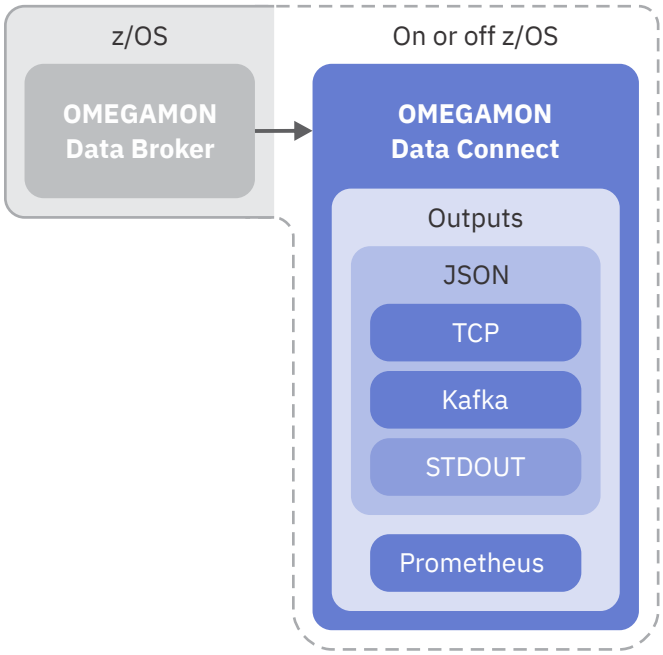

Figure 19. OMEGAMON Data Connect configuration points: input from OMEGAMON Data Broker and various outputs

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:
  input:
    tcp: # Required
      <TCP input parameters>

  output: # At least one output is required
    tcp:
      <TCP 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>

server: # Optional
  <Server parameters>

logging: # Optional
  <Logging parameters>
```

Tip: Use a YAML validator to check that your configuration file conforms to YAML syntax.

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

By default, OMEGAMON Data Connect configuration parameters are stored in the `config/connect.yaml` file in the OMEGAMON Data Connect installation directory.

Tip: To avoid service updates overwriting your edited version of this file, consider setting `spring.config.additional-location` in the OMEGAMON Data Connect startup procedure or script to a file path outside of 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:
```

Parameter descriptions

`connect`
This is the root for parameters that are specific to OMEGAMON Data Connect:

`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 for TCP output.

`kafka`
JSON published to Apache Kafka. You can publish either to a single topic, or to a separate topic for each attribute group (table).

`prometheus`
Prometheus endpoint hosted by OMEGAMON Data Connect.

`stdout`
JSON Lines written to the `stdout` file.

filter
Filters which tables (attribute groups) and which fields (attributes) from those tables to send to the JSON-format outputs: `tcp`, `kafka`, and `stdout`.

event-publisher
Controls aspects of internal OMEGAMON Data Connect processing.

server
Sets Spring Boot server properties.

logging
Sets Spring Boot logging properties.

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 15361 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.

Restarting OMEGAMON Data Connect
If you are running OMEGAMON Data Connect on z/OS, then you can enter an MVS **MODIFY** system command to restart it. Restarting OMEGAMON Data Connect reloads its configuration parameters.

TCP input parameters

OMEGAMON Data Connect TCP input parameters specify how OMEGAMON Data Connect listens for attributes over a TCP network from OMEGAMON Data Broker.

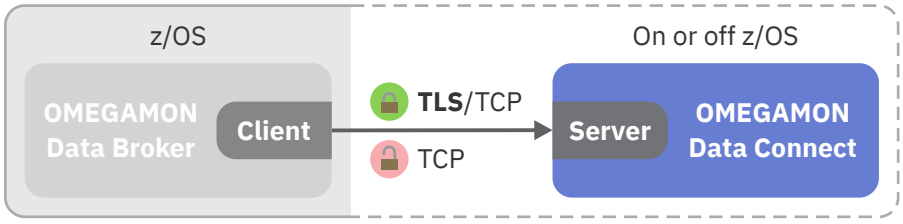

Figure 20. OMEGAMON Data Connect configuration: TCP input

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: boolean
      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
Host name or IP address on which the OMEGAMON Data Connect host listens for data from OMEGAMON Data Broker.

If you run OMEGAMON Data Connect on the same z/OS instance as OMEGAMON Data Broker, then you can specify `localhost` as the host name.

This value must match the OMEGAMON Data Broker parameter `KAY.CIDB.FWD.OM.SINK_HOST`.

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

This value must match the OMEGAMON Data Broker parameter `KAY.CIDB.FWD.OM.SINK_PORT`.

SSL parameters

`connect.input.tcp.ssl`:

```
enabled: boolean
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.

protocol
Protocol to use.

This key is optional. Default: TLS. Recommended: `TLSv1.2`.

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. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore, or *key ring*. Only available if OMEGAMON Data Connect is running on z/OS.

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: 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 PCKS12 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...
```

TCP output parameters

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

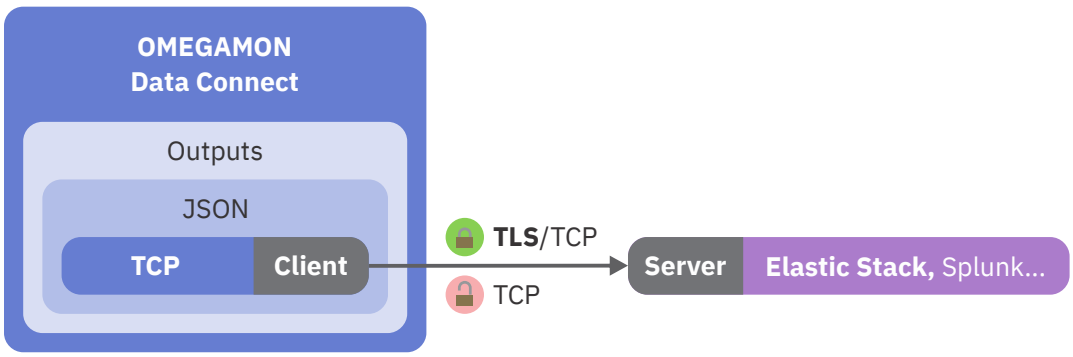

Figure 21. OMEGAMON Data Connect configuration: TCP output

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: boolean # Default at this level: false
      sinks: # One or more sinks (destinations)
        sink_name_1: # Each sink has a unique name of your choice
          enabled: boolean # Default at this level: true
          hostname: string
          port: number
          max-connection-attempts: number # Optional
          retry-interval: seconds # Optional
          ssl: # Optional
            <SSL parameters>
          filter: # Optional output-level filter
            <Filter parameters>
        sink_name_2: # Optional 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 host name 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.ssl`:

```
enabled: boolean
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.

protocol
Protocol to use.

This key is optional. Default: TLS. Recommended: `TLSv1.2`.

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. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore, or *key ring*. Only available if OMEGAMON Data Connect is running on z/OS.

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: localhost # on same z/OS instance as OMEGAMON Data Connect
      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: localhost
      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 PCKS12 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:
    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 which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, Kafka, and STDOUT.

Characteristics of JSON output from OMEGAMON Data Connect

If you need to work directly with the 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.

Kafka output parameters

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

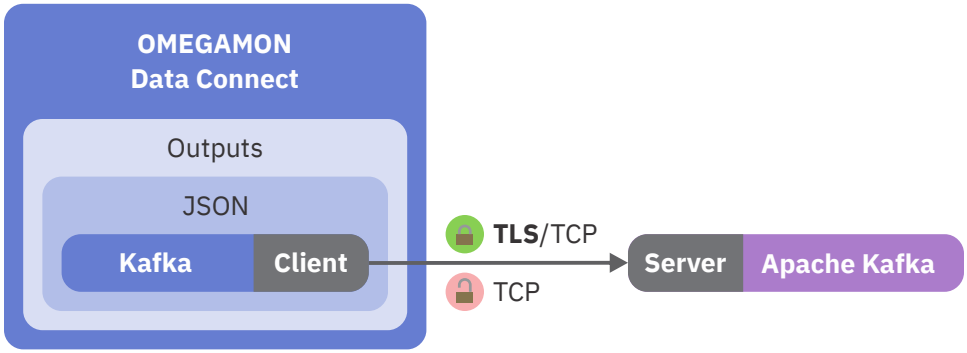

Figure 22. OMEGAMON Data Connect configuration: Kafka output

In this context, OMEGAMON Data Connect is a Kafka *client*.

```
connect:
  output:
    kafka:
      enabled: boolean
      servers: string
      retry-interval: number # Optional. Default: 30 (seconds)
      max-connection-attempts: number # Optional. Default: unlimited
      topic: topic_name # Optional. Default: per-table topics
      topic-prefix: topic_prefix # Optional. Default: odp
      filter: # Optional output-level filter
        <Filter parameters>
      properties: # 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.

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.

retry-interval
The number of seconds to wait between retrying connection to the Kafka servers. This key is optional. Default: 30.

max-connection-attempts
The maximum number of connection attempts. This key is optional. Default: unlimited. If you specify this key, you must specify an integer value; there is no literal value for "unlimited".

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
Kafka client properties, such as SSL parameters. For information about Kafka client properties, see the Apache Kafka documentation.

SSL parameters

Only some Kafka client SSL parameters and allowed values are shown here.

Tip: If a property name contains periods, enclose the name in single or double quotes.

`connect.output.kafka.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
```

For more details on these and other Kafka client SSL parameters, see the Apache Kafka documentation.

Example: Connection without SSL/TLS

The following example sends all attributes to a single Kafka topic, `omegamon-json`.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
```

```
      port: 15379

  output:
    kafka:
      enabled: true
      servers: kafka.example.com:9095
      retry-interval: 10
      max-connection-attempts: 20
      topic: omegamon-json
```

Example: Connection with SSL/TLS

The following example sends attributes to Kafka topics named omegamon.*product.table_name*.

```
connect:
  input:
    tcp:
      enabled: true
      hostname: 0.0.0.0
      port: 15379

  output:
    kafka:
      enabled: true
      servers: kafka1.example.com:9095,kafka2.example.com:9095
      topic-prefix: omegamon
      properties:
        "security.protocol": SSL
        "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
```

Related reference

Filters for JSON-format outputs
You can optionally filter which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, Kafka, and STDOUT.

Characteristics of JSON output from OMEGAMON Data Connect
If you need to work directly with the 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.

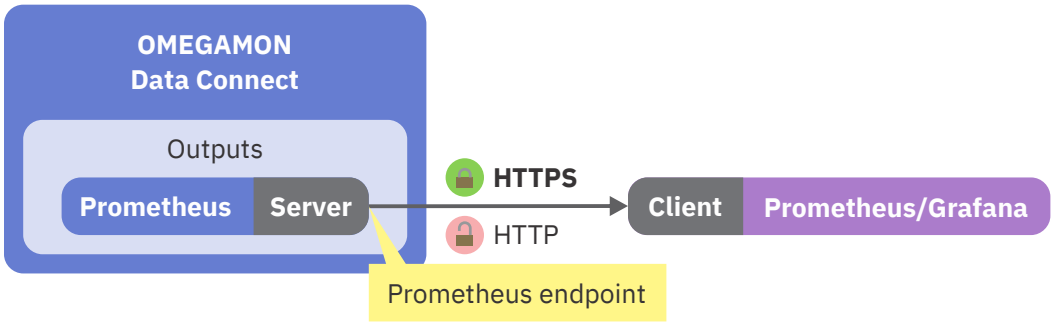

Figure 23. OMEGAMON Data Connect configuration: Prometheus output from an HTTP(S) perspective

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.

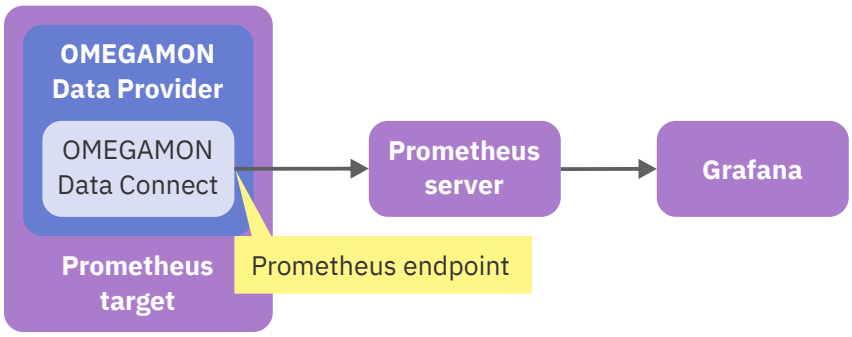

Figure 24. OMEGAMON Data Provider is a Prometheus target

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: boolean
      mappings:
        products:
          kpp: # Example: km5, for the z/OS monitoring agent
            enabled: boolean
              tables:
                table_name:
                  enabled: boolean
                  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 k*pp* (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`.

`tables`
Specifies the tables (attribute groups), and metrics (attributes) from those tables, that OMEGAMON Data Connect publishes to the Prometheus endpoint.

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 attributes to publish as metrics.

`labels`
A list of attribute field names in the table to use as metric labels. Typically, labels refer to *string* attributes, 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.k`*`pp`*`.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 of an attribute in the table. Typically, metrics refer to *numeric* attributes, 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 Spring Boot server properties `server.address` and `server.port` determine the hostname and port of the endpoint URL where OMEGAMON Data Connect publishes Prometheus metrics.

The path component of the endpoint URL is `metrics`.

By default, the endpoint URL uses HTTP, not HTTPS. To use a secure connection (HTTPS), specify `server.ssl` properties.

For example, given the following values:

```
server:
  address: myserver.example.com
  port: 9090
```

the endpoint URL is:

`http://myserver.example.com:9090/metrics`

If you do not specify `server.address` or `server.port`, then the default metrics endpoint URL is:

`http://localhost:9070/metrics`

Metrics expiry

OMEGAMON collects data periodically, according to the historical collection interval that you specify for each attribute group (table).

OMEGAMON Data Connect publishes metrics based on the latest collected data, refreshing metrics at the endpoint as new data arrives.

If a *time series* for a metric has no value in a new interval, then OMEGAMON Data Connect removes the time series from the endpoint. This is known as metrics expiry.

For example, if a metric is labeled by job name, then OMEGAMON Data Connect publishes metrics for a job name only while a corresponding job is running; only for the intervals in which incoming attribute data contains that job name.

Tip: Some 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.

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 25. Example Prometheus text-format output

Related reference

Server parameters
OMEGAMON Data Connect uses the Spring Boot Java framework. The **`server`** key sets Spring Boot server parameters (properties).

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 attributes in JSON Lines format to the `stdout` file.

```
connect:
  output:
    stdout:
      enabled: boolean
      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 which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, Kafka, and STDOUT.

Characteristics of JSON output from OMEGAMON Data Connect
If you need to work directly with the 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 which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, 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.k`*`pp`*`.tables`.

You can filter attributes by product (agent), by table (attribute group), and individually by field name.

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.kafka.filter
```

If you specify a filter, then only attributes enabled by the filter are sent.

If you do not specify a filter, then all attributes from all tables from all products are sent.

Global-level and output-level filters have the same format:

```
filter:
  enabled: boolean
  include: file_path # If specified, the products key is ignored
  products:
    kpp: # Product code
      enabled: boolean
      tables: # Optional. Default: send all tables from this product
        table_name:
          enabled: boolean
```

```
          condition: # Optional. Default: send all records from this table
            enabled: boolean
            expression: SpEL expression
          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 k*pp* 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.k`*`pp`*	No data from this product is sent.
`filter.products.k`*`pp`*`.tables.`*`table_name`*	No data from this table is sent.
`filter.products.k`*`pp`*`.tables.`*`table_name`*`.condition`	No condition is set. All records of this table are sent.

include

Optional. Uses the filter defined in an external *filter include file*.

If you specify an `include` key, then OMEGAMON Data Connect ignores the `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
```

OMEGAMON Data Connect searches for the filter include *file_path* first in the file system, and then in the Java class path.

The *file_path* can be either absolute or relative. When searching the file system, 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.

The OMEGAMON Data Connect JAR file contains an embedded filter include file for Instana. To use the embedded filter for Instana, specify the following file path:

```
filter:
  enabled: true
  include: filters/instana.yaml
```

To use the file that is embedded in the JAR file, ensure that you do not have a `filters/instana.yaml` file in the working directory of your file system. Otherwise, OMEGAMON Data Connect will use the file from your file system instead of the embedded file.

products
Only fields from the specified products (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.

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

You must specify at least one child key under the k*pp* 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)`

Expression using a relational operator	Equivalent expression using a method
	Tip: The `matches` method offers a shorthand for testing alternative values, but uses 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.

An expression can cause a runtime error for several reasons:

Syntax error
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.

Error in expression
OMEGAMON Data Connect checks the expression for structural errors that did not trigger a syntax error.

For example:

- Misspelled field names.
- Attempting to set the value of a read-only field. Typical cause: mistakenly using a single equal sign (=) to compare for equality instead of the correct two consecutive equal signs (==) .

If the expression fails this check, then OMEGAMON Data Connect performs the following actions:

1. Reports error message KAYC0048E, followed by the error details.
2. Reports informational message KAYC0056I.
3. Discards the record currently being processed.
4. Disables the table in outputs that use this expression.

   If this expression is in the global-level filter, then OMEGAMON Data Connect disables the table in all outputs that use the global-level filter. Outputs that specify their own (output-level) filter are unaffected.

   If the expression is in an output-level filter, then OMEGAMON Data Connect disables the table in that output only. All other outputs are unaffected.

Error caused by value in expression
Some errors occur only when a field in the expression has a particular value. For example, if the expression uses an integer field as the denominator in a division operation, then a divide-by-zero error occurs only if the value of that field is zero. For nonzero denominators, the division operation succeeds and the expression resolves to true or false.

OMEGAMON Data Connect reports these errors in message KAYC0031W.

Except for that message, OMEGAMON Data Connect behaves as if the expression returned a false value: it discards the record, and continues to process subsequent records that use this expression.

Tip:

- If an expression refers to a field that might not be in every record, then, to avoid throwing a null pointer exception at runtime, 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 attribute field names to send from this table.

OMEGAMON Data Connect always sends the common fields `write_time` 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 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 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 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 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 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 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 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 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: Instana filter embedded in OMEGAMON Data Connect

The following TCP output uses the filter include file for Instana that is embedded in the OMEGAMON Data Connect JAR file.

```
connect:
  output:
    tcp:
      enabled: true
      sinks:
        instana:
          hostname: instana.example.com
          port: 5046
          filter:
            include: filters/instana.yaml
```

For details on sending attributes to Instana, see the Instana documentation.

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. 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
```

Related reference

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

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

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

STDOUT output parameters

OMEGAMON Data Connect STDOUT output parameters specify whether to write attributes 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
```

queue-capacity
The maximum number of records that OMEGAMON Data Connect stores in its queue.

This value specifies a number of records. The corresponding amount of storage depends on the details of each record.

Default: `0`, meaning unbounded; no maximum limit.

If you set a limit, and the queue exceeds the limit, then OMEGAMON Data Connect throttles incoming records. OMEGAMON Data Connect blocks (stops accepting) incoming records until the queue no longer exceeds the limit.

Server parameters

OMEGAMON Data Connect uses the Spring Boot Java framework. The **server** key sets Spring Boot server parameters (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 server address, port, and SSL parameters.

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 parameters>
```

address
Host name or IP address on which to listen for requests. Default: `localhost`.

port
Port number on which to listen for requests. Default: `9070`.

SSL parameters

SSL parameters 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: boolean
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 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.

protocol
Protocol to use.

This key is optional. Default: TLS. Recommended: `TLSv1.2`.

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 four (4) 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. Examples:

JKS
Java keystore.

PKCS12
Public-Key Cryptography Standards (PKCS) #12.

JCERACFKS
Java Cryptography Standards (JCE) RACF keystore, or *key ring*. Only available if OMEGAMON Data Connect is running on z/OS.

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.

Logging parameters

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.

Allowed values: `ERROR`, `WARN`, `INFO` (default), `DEBUG`, `TRACE`.

For details of logging levels and their meanings, see the Spring Boot documentation.

Methods for setting the logging level

Use any of the following methods to set the logging level:

- Set the `logging.level.com.rocketsoft` property value in the YAML configuration file, `connect.yaml`. See the example at the end of this topic.
- If you are using the supplied sample KAYCONN JCL procedure member to run OMEGAMON Data Connect, either:
  - Set the LOGLEVEL symbolic parameter in the procedure JCL:

    ```
    SET LOGLEVEL='ERROR'
    ```

  - When starting the JCL procedure, specify the logging level on the MVS **START** system command. For example:

    ```
    S KAYCONN,LOGLEVEL=WARN
    ```

- If you are using the supplied sample `connect` shell script to start OMEGAMON Data Connect, specify the `logging.level.com.rocketsoft` property value as a command-line option. For example:

  ```
  connect --logging.level.com.rocketsoft=TRACE
  ```

Example

```
logging:
  level:
    com:
      rocketsoft: INFO
```

Troubleshooting

To diagnose and correct problems that you experience with OMEGAMON Data Provider, first examine the messages from the related components.

Next, check the common issues described here.

Finally, if you cannot resolve the problem, gather diagnostic information before contacting IBM Software Support.

Tip:

- 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.
- 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 "Overview of configurable parts" on page 17.
- 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.

Related reference

Messages
Each component of 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 attributes are not arriving at a destination analytics platform, but there are no obvious errors, then use these messages as a checklist to diagnose the problem.

Gathering diagnostic information

Before you report a problem with OMEGAMON Data Provider to IBM Software Support, you need to gather the appropriate diagnostic information.

The following procedure lists the information that you need to gather and then send to IBM Software Support to help diagnose a problem.

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)
   Collection configuration

   PARMLIB(ZWESIS*xx*)
   Zowe cross-memory server configuration, containing OMEGAMON Data Broker configuration parameters

   config/connect.yaml
   OMEGAMON Data Connect configuration
3. Gather the complete job log and any dumps from each of the z/OS address spaces involved.
   - 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?

Common issues

Before contacting IBM Software Support, check for these common issues.

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'.

No KPQH037I or KPQH038I message for a table

Symptoms

The address space where a collection task is running (typically, the monitoring agent address space) is missing an expected KPQH037I or KPQH038I message, or both, for a table (attribute group).

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 Network Monitor
   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 OMEGAMON runtime environment that collects attributes.

Adding more 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

OMEGAMON Data Provider collection configuration parameters

Collection tasks use OMEGAMON Data Provider collection configuration parameters to 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 attributes are 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

Messages

Each component of OMEGAMON Data Provider writes messages that describe activity or errors.

Message location and prefix by component

Component	Message location	Message prefix
Collection tasks	RKLVLOG output data set of the corresponding job. For example, for attributes from IBM Z OMEGAMON Monitor 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 ZWE. For descriptions of ZWE messages, see 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 concepts

Troubleshooting
To diagnose and correct problems that you experience with OMEGAMON Data Provider, first examine the messages from the related components.

Expected messages

These are the normal messages that you should expect from each component involved in OMEGAMON Data Provider. If attributes are 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 flow of attributes: from collection tasks, 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 sources, 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 plugin.

Collection tasks

The RKLVLOG output data set of each monitoring 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 12.

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 "No KPQH037I or KPQH038I message for a table" on page 91.
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 "No KPQH037I or KPQH038I message for a table" on page 91.

OMEGAMON Data Broker

The SYSPRINT output data set of the Zowe cross-memory server job that runs OMEGAMON Data Broker (for example, job name ZWES*x*) 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.
KAYB0009I Init step 'Load CIDB parameters' done KAYB0009I Init step 'CIDB global area initialization' done	Normal initialization messages.

Message	Description
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	
KAYB0036I Store '*store_name*' 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, **MODIFY** commands.
KAYC0035I Build: *build_identifier*	Reports the OMEGAMON Data Connect build identifier.
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 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 output thread [*sink_name*] {host: *hostname*, port: *port*} KAYC0010I Connecting to *hostname*:*port* KAYC0011I Connected to *hostname*:*port* ...	Normal initialization messages for each sink.

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.

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 output thread [*sink_name*] {host: *hostname*, port: *port*}	For each TCP output sink.

Related concepts

Troubleshooting

To diagnose and correct problems that you experience with OMEGAMON Data Provider, first examine the messages from the related components.

Related information

No KPQH037I or KPQH038I message for a table

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

OMEGAMON Data Provider collection configuration parameters
Collection tasks use OMEGAMON Data Provider collection configuration parameters to 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

No KPQH037I or KPQH038I message for a table

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

No KPQH037I or KPQH038I message for a table

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 ZWE. For descriptions of ZWE messages, see 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.

System action:
OMEGAMON Data Broker initialization continues. However, this subsystem will not 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 **Store '*store_name*' 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 **Store *store_name* 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 **Store *store_name* 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_name*' 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_name*' 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_name*'**

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_name*' 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.

KAYBN000E **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 **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 **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 **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 **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.

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.

KAYBN005E **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 include the host name and port on which OMEGAMON Data Connect is listening.

KAYBN006E **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 **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 **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 **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 **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 **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 include the host name and port on which OMEGAMON Data Connect is listening.

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 *retryInterval* 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
This is the first time, either since starting or since its configuration was refreshed by a **MODIFY** command, that this instance of OMEGAMON Data Connect has received data for this table. "Mapping class" refers to code in OMEGAMON Data Connect that transforms 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 *hostname*:*port***

Explanation:
OMEGAMON Data Connect is attempting to connect to the TCP output destination specified by a `connect.output.tcp` configuration parameter.

System action:

None.

User response:
None required.

KAYC0011I Connected to *hostname*:*port*

Explanation:
OMEGAMON Data Connect has successfully connected to the TCP output destination specified by a `connect.output.tcp` configuration parameter.

System action:
None.

User response:
None required.

KAYC0012E Error connecting to *hostname*:*port*

Explanation:
OMEGAMON Data Connect could not connect to the output TCP destination specified by a `connect.output.tcp` configuration parameter.

System action:
OMEGAMON Data Connect continues, but does not send output to that destination.

User response:
Check that the destination *hostname*:*port* is listening for JSON Lines over TCP from OMEGAMON Data Connect.

KAYC0013E Maximum reconnection attempts (*maxConnectionAttempts*) to *host*:*port* reached. TCP output service is stalled

Explanation:
OMEGAMON Data Connect could not reconnect to the output TCP destination specified by a `connect.output.tcp` configuration parameter.

System action:
OMEGAMON Data Connect continues running, but does not send output to that destination.

User response

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.maxConnectionAttempts`.
3. Restart OMEGAMON Data Connect.

KAYC0014E I/O error writing to peer socket: *details*. Reconnection will be attempted in *retryInterval* 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.retryInterval`.
3. Restart OMEGAMON Data Connect.

KAYC0015E Error creating JSON

Explanation:
OMEGAMON Data Connect encountered an error creating JSON for output to TCP.

System action:
The attribute is not sent to the TCP 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 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.tables.`*`table_name`*`.metrics.type` specified an unhandled Prometheus metric type:

```
prometheus:
      enabled: true
      endpoint: metrics
      tables:
        table_name:
          metrics:
            - name: field_name
              help: metric_help
              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.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 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.

KAYC0030I Restarting server

Explanation
OMEGAMON Data Connect is restarting.

For example, OMEGAMON Data Connect received an MVS system **MODIFY** command requesting a restart to refresh its configuration.

System action:
None.

User response:
None required.

KAYC0031W Event publication error: *details*

Explanation:
OMEGAMON Data Connect encountered an error publishing an event.

System action:
Depends on the details following the message.

User response:
Review the error details following this message. If you cannot resolve the issue, contact IBM Software Support.

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
This is the first time, either since starting or since its configuration was refreshed by a **MODIFY** command, that 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 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 table 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

System action:
None.

User response:
None required.

Related reference

Filters for JSON-format outputs
You can optionally filter which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, 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 output thread [*sink_name*] {host: *hostname*, port: *port*}**

Explanation:
OMEGAMON Data Connect is starting a thread for the TCP output specified in the configuration parameters by the YAML key `connect.output.tcp.sinks.sink_name`.

System action:
None.

User response:
None required.

KAYC0043I **Stopping TCP output thread [*sink_name*] {host: *hostname*, port: *port*}**

Explanation:
OMEGAMON Data Connect is stopping the thread for the TCP output specified in the configuration parameters by the YAML key `connect.output.tcp.sinks.sink_name`.

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

Attribute dictionary
OMEGAMON Data Connect includes an attribute dictionary 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:
Edit the configuration parameters, and then restart OMEGAMON Data Connect. If you cannot resolve the issue, contact IBM Software Support.

KAYC0048E **Error checking condition**

Explanation

The OMEGAMON Data Connect configuration parameters specify a filter condition. The condition expression contains a structural error that did not trigger a syntax error.

For example:

- Misspelled field names.
- Attempting to set the value of a read-only field. Typical cause: mistakenly using a single equal sign (=) to compare for equality instead of the correct two consecutive equal signs (==) .

System action

OMEGAMON Data Connect performs the following actions:

1. Discards the record currently being processed.
2. Disables the table in outputs that use this filter.
3. Reports message KAYC0056I.

User response:
Edit the expression, and then restart OMEGAMON Data Connect. If you cannot resolve the issue, contact IBM Software Support.

Related reference

Filters for JSON-format outputs
You can optionally filter which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, Kafka, and STDOUT.

KAYC0049E **Error publishing to Kafka**

Explanation:
OMEGAMON Data Connect encountered an error attempting to send attributes 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:
Remove the `include` parameter from the filter include file. 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 KAYC0048E, which reports an error in a filter condition expression. As a result of that error, OMEGAMON Data Connect disables (stops sending records of) the table in outputs that use this filter.

System action

OMEGAMON Data Connect continues processing, but disables the table in outputs that use this filter.

If this expression is in the global-level filter, then OMEGAMON Data Connect disables the table in all outputs that use the global-level filter. Outputs that specify their own (output-level) filter are unaffected.

If the expression is in an output-level filter, then OMEGAMON Data Connect disables the table in that output only. All other outputs are unaffected.

User response:
No response required for this message. See the response for message KAYC0048E.

Related reference

Filters for JSON-format outputs

You can optionally filter which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, Kafka, and STDOUT.

Reference

Monitoring agents supported by OMEGAMON Data Provider

OMEGAMON Data Provider processes attributes from several OMEGAMON monitoring agents.

Table 6. Monitoring agents supported by OMEGAMON Data Provider, with links to attributes documentation

Product code	Monitored subsystem	Monitoring agent	Monitoring agent: minimum version supported
kc5	CICS	IBM Z OMEGAMON for CICS Previous name, before V5.6: IBM Z OMEGAMON for CICS on z/OS	5.5
kgw	CICS TG	IBM Z OMEGAMON for CICS TG Previous name, before V5.6: 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 V5.5: 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 for JVM on z/OS	5.5
kmq	MQ	IBM OMEGAMON for Messaging on z/OS	7.5
kn3	Network	IBM Z OMEGAMON Network Monitor	5.6
ks3	Storage	IBM OMEGAMON for Storage on z/OS	5.5
km5	z/OS	IBM Z OMEGAMON Monitor for z/OS	5.6

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

OMEGAMON Data Provider collection configuration parameters

Collection tasks use OMEGAMON Data Provider collection configuration parameters to 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 which attributes to send to the JSON-format outputs of OMEGAMON Data Connect: TCP, 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.

Attribute dictionary

OMEGAMON Data Connect includes an attribute dictionary in a set of YAML files.

The dictionary describes the product codes, table names, and attribute field names that you can specify in OMEGAMON Data Provider collection configuration parameters and OMEGAMON Data Connect configuration parameters.

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: The file for the con table is named con**t**.yaml, with a t appended to the table name. Windows reserves con for a device file and does not allow con.yaml as a file 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.
```

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.

Format of OMEGAMON attribute names	Example rules for field names
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 Monitor 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`
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`

OMEGAMON attribute name	OMEGAMON Data Provider field name
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 Monitor 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

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 includes the following fields in output for all products, for all attribute tables. These fields are sometimes referred to as *common fields*. These fields are included regardless of any field-level filter.

interval_seconds
The collection interval of the historical collection, in seconds.

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.

write_time
Timestamp when the data was created on z/OS by the OMEGAMON collection task, before it was forwarded to OMEGAMON Data Broker.

km5: z/OS monitoring agent

OMEGAMON Data Connect introduces the following fields to data from IBM Z OMEGAMON Monitor 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
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 the 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.

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 119.

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 attributes in JSON Lines format over a TCP network.

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

STDOUT output parameters
OMEGAMON Data Connect STDOUT output parameters specify whether to write attributes in JSON Lines format to the `stdout` file.

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.