FUJITSU Software
Interstage Big Data
Complex Event Processing Server V1.1.0

User's Guide

Linux(64)

J2UL-1665-02ENZ0(00)
February 2014

Preface

Purpose of this document

This manual provides an overview of the features of Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP"). It also describes the operations required during installation and application development, and the operation and maintenance of BDCEP.

Intended readers

This manual is intended for users who are considering installing, operating, and developing applications that use the Complex Event Processing feature of BDCEP.

Structure of this document

This document is structured as follows:

Chapter 1 Overview

Provides an overview of BDCEP.

Chapter 2 Features Provided

Describes the features provided by BDCEP.

Chapter 3 System Configuration and Design

Describes system design for installing BDCEP, such as what kind of information system to build by installing the product, and how to design the operation form.

Chapter 4 Installation and Setup

Describes the software requirements and resources required to install BDCEP, as well as how to install and uninstall it.

Also, explains the setup of BDCEP (that is, how to create an environment for building the system).

Chapter 5 Development

Describes how to develop applications to run on BDCEP.

Chapter 6 Operation and Maintenance

Describes how to operate and manage built systems, and how to manage BDCEP.

Chapter 7 Extended System Operations

Describes how to operate BDCEP using multiple servers.

Chapter 8 Command Reference

Describes the commands of BDCEP.

Chapter 9 Definition File Reference

Describes the definition files handled by BDCEP.

Glossary

Explains the terminology used for BDCEP.

Trademarks

- Adobe, Adobe Reader, and Flash are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

- Linux is a registered trademark of Linus Torvalds.

- Red Hat, RPM, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

Export restrictions

If this document is to be exported or provided overseas, confirm legal requirements for the Foreign Exchange and Foreign Trade Act as well as other laws and regulations, including U.S. Export Administration Regulations, and follow the required procedures.

Copyright

February 2014: Second edition
October 2012: First edition

Contents

Chapter 1 Overview

This chapter provides an overview of the features provided by Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP").

1.1 What is Interstage Big Data Complex Event Processing Server?

BDCEP is software that analyzes and assesses massive volumes of event data in real time.

In recent years, there has been a growing demand from companies wanting to use ever-changing event data generated in massive volumes, such as location information sent from smart phones and machines' operation logs, in order to leverage their business activities.

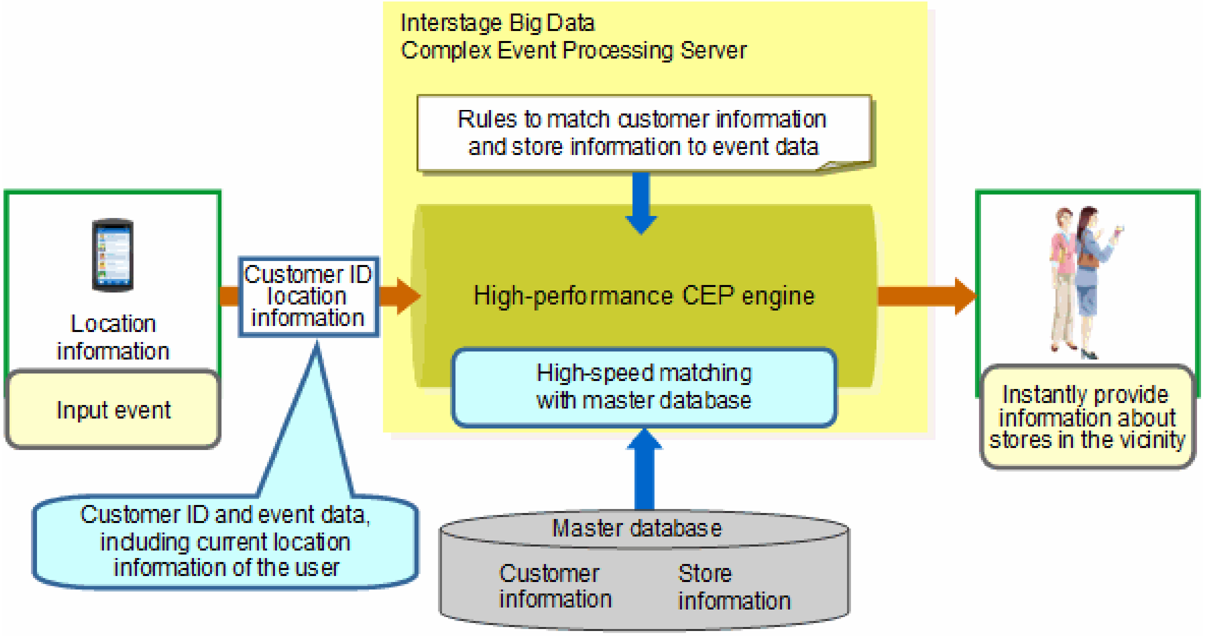
The need to process these kinds of event data in real time has drawn attention to the CEP (Complex Event Processing) technique, which analyzes and assesses massive volumes of data with faster response times than ever before.

BDCEP includes a high-performance complex event processing engine (hereafter referred to as the "high-performance CEP engine") which integrates a unique high-speed filter processing technique with the complex event processing technique so suitable for processing massive volumes of event data. This engine provides enhanced processing performance and convenience to support real-time use of massive volumes of event data in corporate systems.

Some scenarios for using BDCEP are described below.

Provide real-time services by utilizing location information

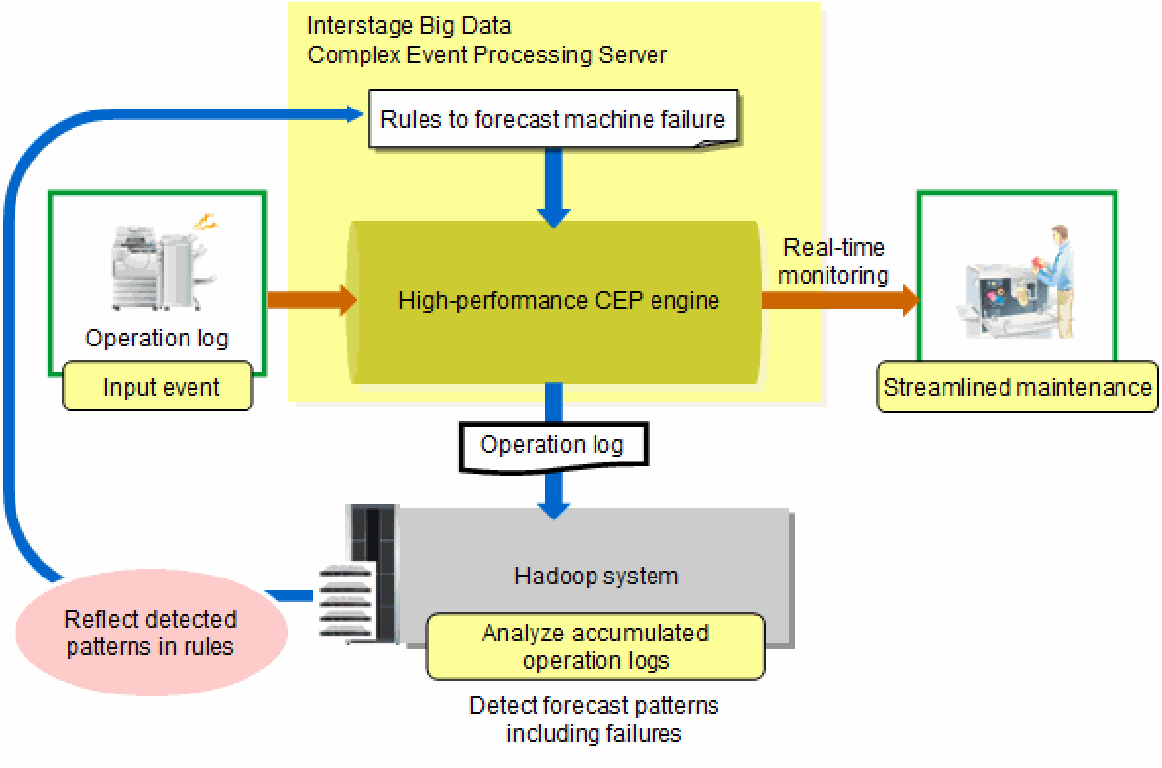
BDCEP allows high-speed matching of real-time customer location information with information registered in the master data, such as customer information and store information. This allows companies to instantly provide services to suit the attributes of customers, such as "provide store coupons to people visiting the vicinity of a store, in real time".


Interstage Big Data
Complex Event Processing Server
Rules to match customer information and store information to event data
Location information
Input event
Customer ID location information
High-performance CEP engine
High-speed matching with master database
Instantly provide information about stores in the vicinity
Customer ID and event data, including current location information of the user
Master database
Customer information
Store information

Improve service by monitoring the operation status of sold products

BDCEP allows real-time monitoring the fault prediction of hardware sold to customers, by collecting the operation logs of hardware. This enhances machine availability by allowing preventive maintenance to be performed, which previously may have been impossible in periodic maintenance due to cost or other factors.

The operation logs collected by BDCEP can also be accumulated and analyzed in a Hadoop system, which allows the detection of more refined prediction patterns. Reflecting these patterns in the complex event processing rules allows the implementation of more efficient maintenance services.


Interstage Big Data
Complex Event Processing Server
Rules to forecast machine failure
Operation log
Input event
High-performance CEP engine
Real-time monitoring
Streamlined maintenance
Operation log
Hadoop system
Analyze accumulated operation logs
Detect forecast patterns including failures
Reflect detected patterns in rules

1.2 Product Features

This section explains the features of BDCEP, which include the following:

- High-performance CEP Engine
- Simple Rule Description
- Simple Collaboration with External Systems

1.2.1 High-performance CEP Engine

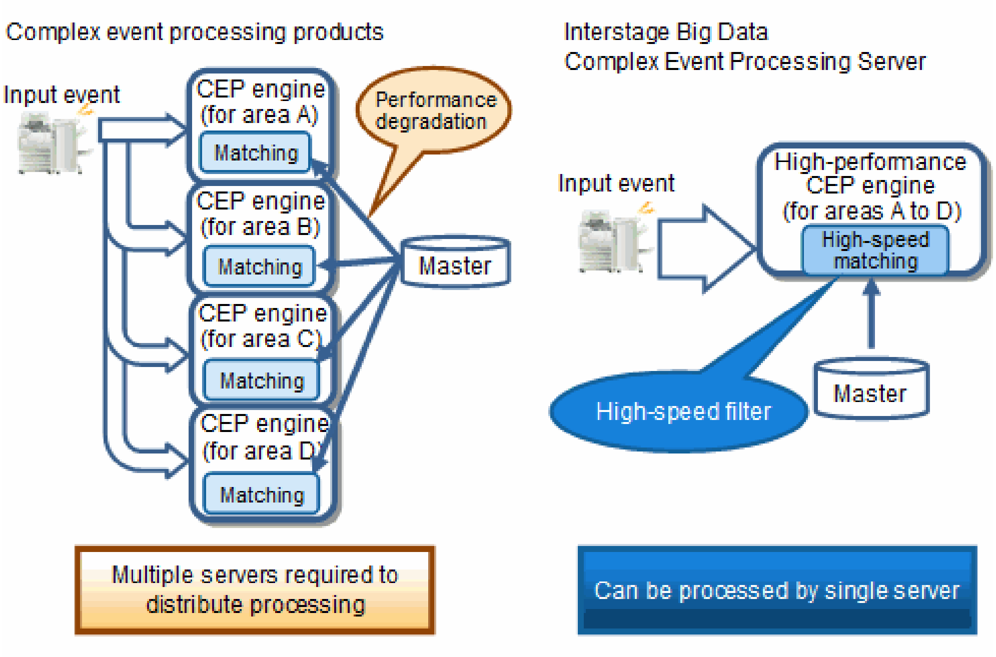
The inclusion of the high-performance CEP engine allows massive volumes of event processing by one server.

In conventional complex event processing products, performance dramatically declines if massive volumes of data are accessed in external master data. To overcome that and ensure performance, multiple servers are provided to distribute processing.

BDCEP uses the unique technique of high-speed filters to allow high-speed matching of input events with master data.

This results in substantially improved performance compared with the conventional complex event processing engine, and allows the required number of events to be processed by one server, with no decline in performance.

Figure 1.1 Comparison when massive volumes of events are processed



1.2.2 Simple Rule Description

Rules must be set in advance in order to execute complex event processing.

BDCEP uses the two description formats below in order to allow rule definitions to be created in a flexible way, according to their purpose.

SQL-type format

This format is based on the database query language SQL.

It is suitable for rules with complex conditional branching or event matching processes.

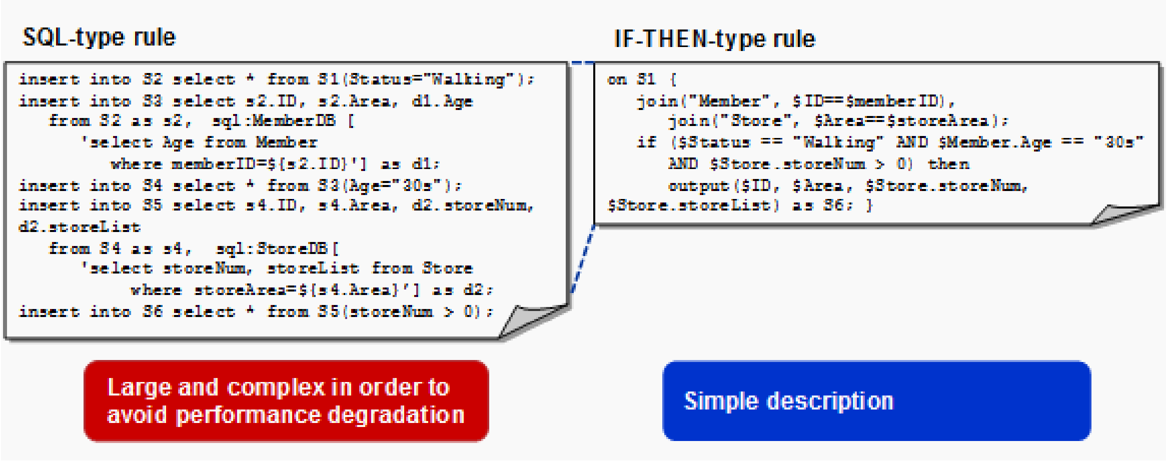
IF-THEN-type format

This format uses the "IF (condition) THEN (process)" structure.

It is suitable for rules that are simpler and easier to understand than SQL-type ones - for example, rules that describe processes such as event filtering, or matching and joining events with master data.

This format also allows simple description of processes that, in conventional SQL-type format, tend to become large and complex, since they need to avoid performance degradation that might be caused by costly join operations.

Figure 1.2 Comparison between an SQL-type rule and an IF-THEN-type rule describing a join with master data



SQL-type rule

```
insert into S2 select * from S1(Status="Walking");
insert into S3 select s2.ID, s2.Area, d1.Age
   from S2 as s2,  sql:MemberDB [
      'select Age from Member
         where memberID=${s2.ID}'] as d1;
insert into S4 select * from S3(Age="30s");
insert into S5 select s4.ID, s4.Area, d2.storeNum,
d2.storeList
   from S4 as s4,  sql:StoreDB[
      'select storeNum, storeList from Store
           where storeArea=${s4.Area}'] as d2;
insert into S6 select * from S5(storeNum > 0);
```

IF-THEN-type rule

```
on S1 {
   join("Member", $ID==$memberID),
      join("Store", $Area==$storeArea);
   if ($Status == "Walking" AND $Member.Age == "30s"
      AND $Store.storeNum > 0) then
      output($ID, $Area, $Store.storeNum,
$Store.storeList) as S6; }
```

1.2.3 Simple Collaboration with External Systems

1.2.3.1 Distributed Cache Collaboration (Terracotta Collaboration)

BDCEP can access data in a distributed cache stored in the in-memory data management software Interstage Terracotta BigMemory Max (hereafter referred to as Terracotta)(*1).

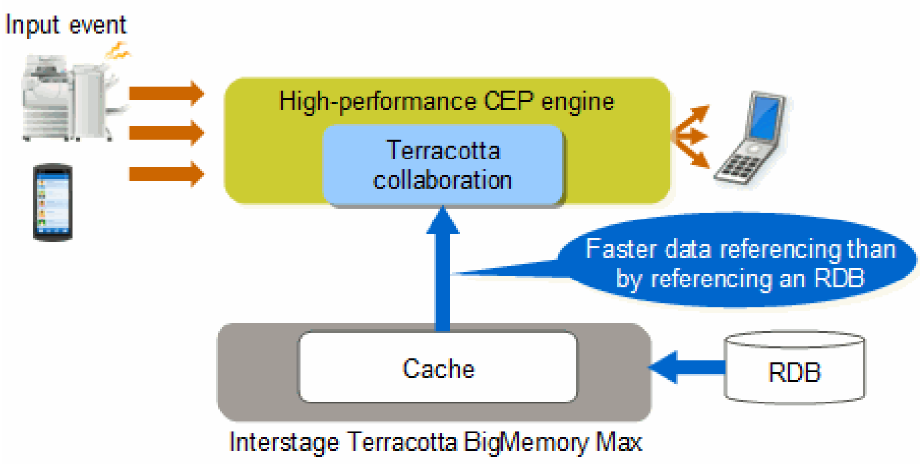
This allows external data to be referenced more rapidly than a relational database (RDB). It also enables cache data to be added, updated, or deleted when an event occurs.

Information

(*1) Interstage Terracotta BigMemory Max

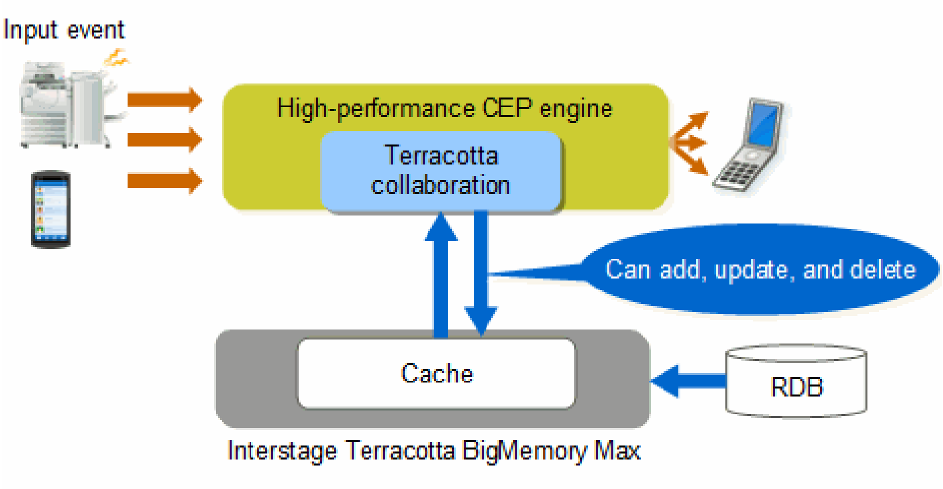
This is in-memory data management software that manages a terabyte or more of business data in server memory and enables ultra high-speed and stable access to business data.

Figure 1.3 Terracotta collaboration (1)



Input event
High-performance CEP engine
Terracotta collaboration
Faster data referencing than by referencing an RDB
Cache
RDB
Interstage Terracotta BigMemory Max

Figure 1.4 Terracotta collaboration (2)


Input event
High-performance CEP engine
Terracotta collaboration
Can add, update, and delete
Cache
RDB
Interstage Terracotta BigMemory Max

1.2.3.2 Hadoop Collaboration

Collaboration with a Hadoop system allows input events and events output by complex event processing to be accumulated in a Hadoop server.

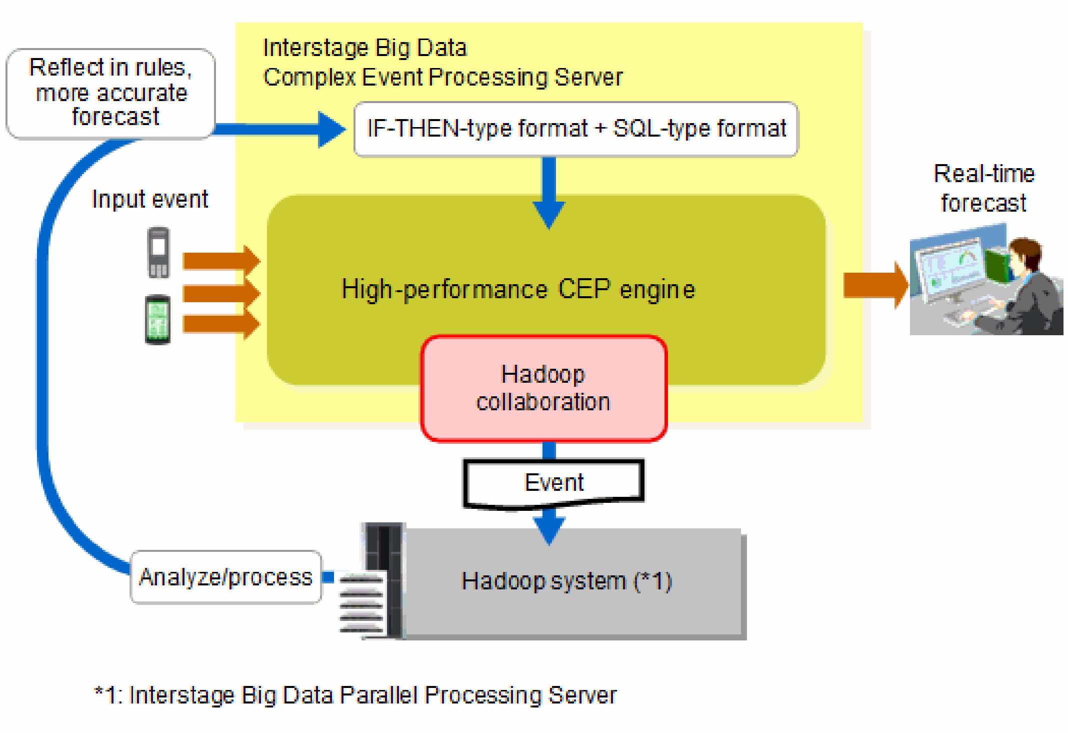
Analysis and processing of the accumulated events by the Hadoop system allows long-term trend analysis of the events, and this can be harnessed for purposes that include business improvement and developing more accurate rules.

The Big Data analysis utilization software known as Interstage Big Data Parallel Processing Server (hereafter, referred to as "BDPP" - see information below) can be used in a Hadoop system.

Information

Based on Apache Hadoop, Interstage Big Data Parallel Processing Server is a Big Data support software on corporate systems that integrates Fujitsu proprietary techniques to further improve processing performance and reliability.

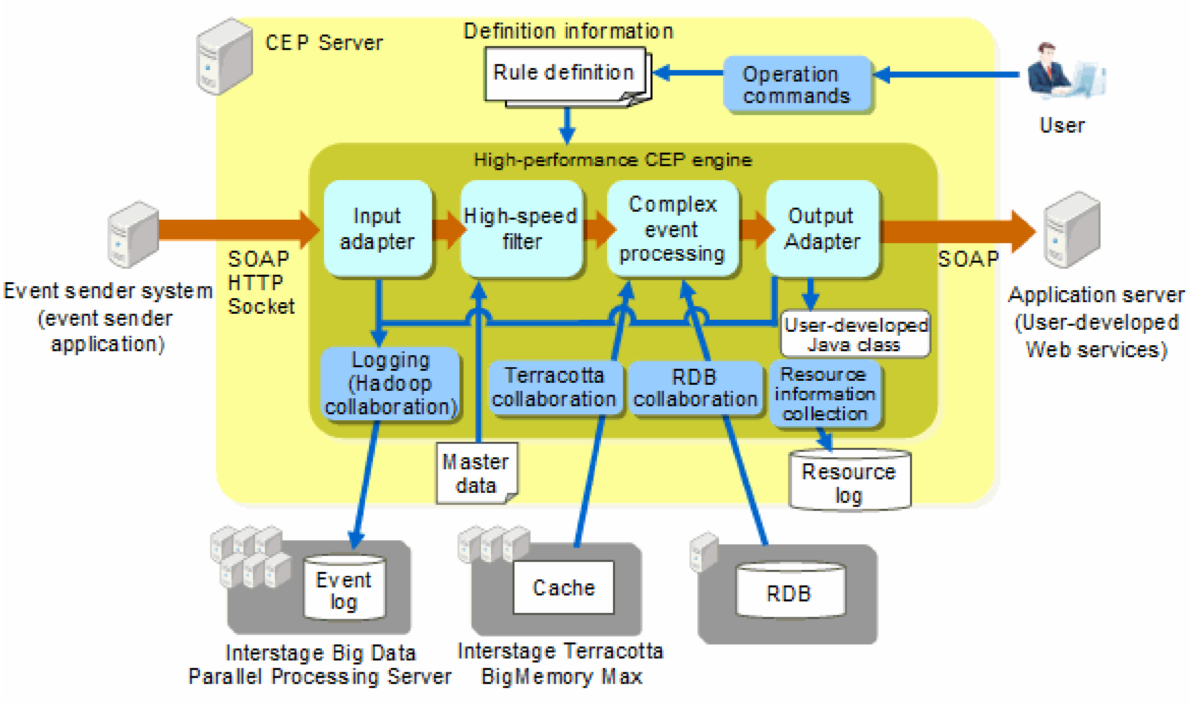
Figure 1.5 Hadoop collaboration


Reflect in rules, more accurate forecast
Interstage Big Data Complex Event Processing Server
IF-THEN-type format + SQL-type format
Input event
High-performance CEP engine
Hadoop collaboration
Real-time forecast
Event
Analyze/process
Hadoop system (*1)
*1: Interstage Big Data Parallel Processing Server

1.3 Overview of Features Provided

This section provides an overview of the features provided by BDCEP (refer to "Chapter 2 Features Provided" for information on each one).

Figure 1.6 List of features provided


CEP Server
Definition information
Rule definition
Operation commands
User
High-performance CEP engine
Input adapter
High-speed filter
Complex event processing
Output Adapter
Event sender system (event sender application)
SOAP
HTTP
Socket
SOAP
Application server (User-developed Web services)
User-developed Java class
Logging (Hadoop collaboration)
Terracotta collaboration
RDB collaboration
Resource information collection
Master data
Resource log
Event log
Cache
RDB
Interstage Big Data Parallel Processing Server
Interstage Terracotta BigMemory Max

1.3.1 Features of the CEP Engine

Input adapter

Provides three communication methods: SOAP, HTTP, and Socket.

It can connect with input sources such as sensors, smartphones, and SOA systems.

Logging (Hadoop collaboration)

Allows input events and events output by complex event processing to be recorded.

When developing rules, the output destination of logging can be a Hadoop system (Hadoop collaboration), or a CEP Server.

Outputting a log to a Hadoop system allows events to be analyzed in it, and this can be harnessed in monitoring event trends and adding or modifying rules.

High-speed filter

Allows input events to be processed rapidly with the pre-registration of a "rule definition" that describes how to filter input events and perform high-speed matching of events with master data.

Complex event processing

Allows continuously generated events to be analyzed and assessed in real time with the pre-registration of a "rule definition" that describes detection patterns for input events.

External data access

Allows external data to be referenced from complex event processing.

Terracotta collaboration

Allows cache data stored in Terracotta to be referenced as external data. It also enables cache data to be added, updated, or deleted when an event occurs.

RDB collaboration

Allows data registered in a relational database to be referenced as external data.

Output adapter

Allows events output by complex event processing to be sent to a user-developed Web service by using the SOAP listener, and to be processed by a user-developed Java class by using the custom listener.

There are also the logging listener (which can be used to store the events output by complex event processing in a Hadoop system), and the debug log listener (which can be used to output the results of complex event processing to a log, for debugging purposes).

1.3.2 Features for Development and Operating Environments

Operation commands

BDCEP provides commands that perform various operations on the CEP engine.

1.3.3 Features for Status Monitoring

Resource log output

This feature logs the CEP engine's resource usage, such as the amount of memory used and the number of input-output events.

The resource log can be used to harness this information in operation and maintenance applications for BDCEP, such as tuning.

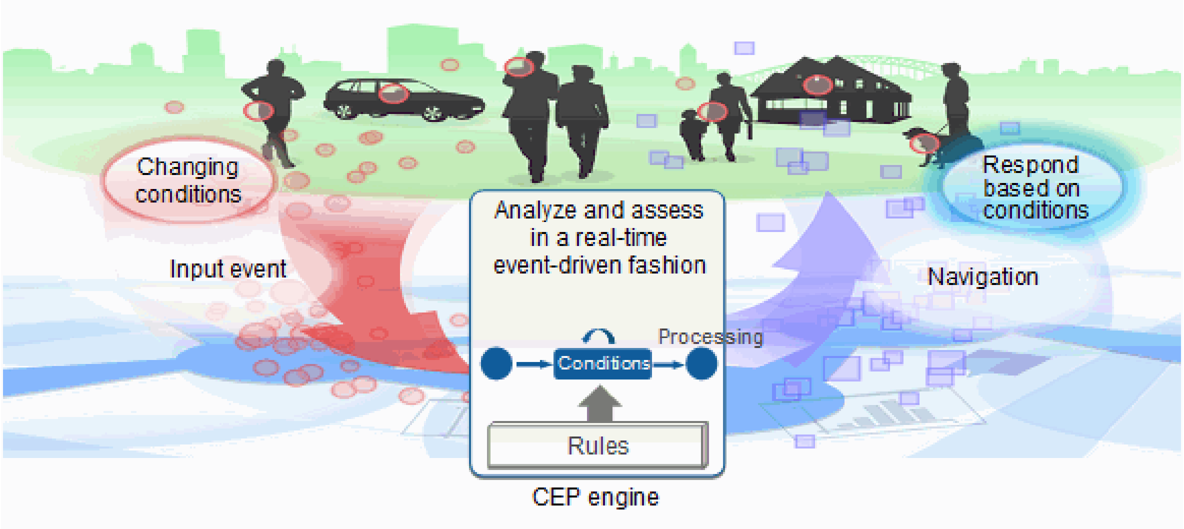
1.4 What is Complex Event Processing?

This section explains complex event processing.

1.4.1 Complex Event Processing

Complex event processing is a technique that analyzes and assesses continuously sent massive volumes of events rapidly and in real time, according to pre-defined rules.

Figure 1.7 Overview of the Complex Event Processing


Changing conditions
Input event
Analyze and assess in a real-time event-driven fashion
Processing
Conditions
Rules
CEP engine
Respond based on conditions
Navigation

Complex event processing has the following features:

Real-time processing

In complex event processing, data input from outside is processed in memory as it is, so massive volumes of input data can be processed more rapidly.

As a result, a response can be returned immediately even in ever-changing conditions.

Generally speaking, the throughput and latency (see information below) that can be achieved using complex event processing are as follows:

- Throughput based on rules: Several tens of thousands to several million events/second
- Processing latency: Several microseconds to several milliseconds

Information

In complex event processing, latency is the time elapsed between an event input and its output.

No program development required

The kinds of input events and processes to be processed by complex event processing are described in rules.

No particular program development is required.

By simply changing a rule, you can change the processing content (in order to change the event pattern to be detected, for example).

Chapter 2 Features Provided

This chapter explains the features provided by Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP").

The features are as follows:

- Main features
    - Input adapter
    - Logging
    - High-speed filter
    - Complex event processing
    - External data access
    - Output adapter
- Operation features
    - Operation commands
    - Engine log
    - Resource log
    - Cluster service

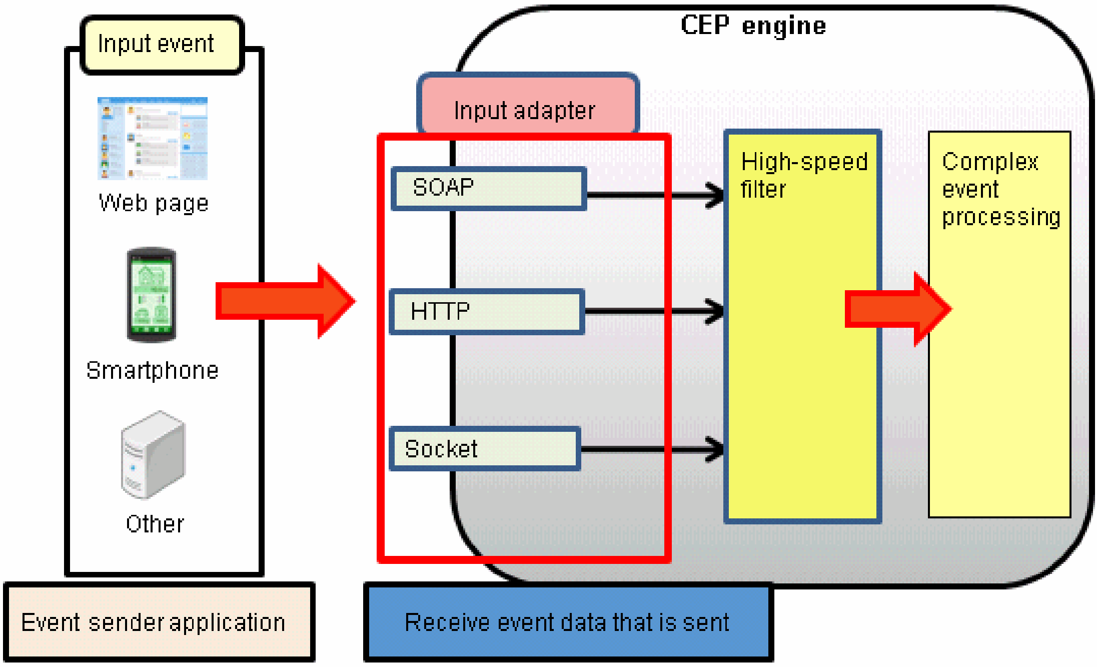
2.1 Input Adapter

The input adapter is a feature that receives events sent from input sources such as sensors, smartphones, and SOA systems.

BDCEP provides the following three input adapters:

- SOAP adapter
- HTTP adapter
- Socket adapter

Figure 2.1 Example of the input adapter



2.1.1 SOAP Adapter

Feature details

The SOAP adapter allows SOAP messages to be received using SOAP communication.

The SOAP adapter extracts event data in XML or CSV format from the received SOAP messages and passes it to the high-speed filter.

Usage scenario

The SOAP adapter is used if event data is to be received from a system that allows SOAP communication, such as an SOA system.

Refer to Chapter 3, "Input Adapter Reference" in the *Developer's Reference* for examples of SOAP messages and for samples of event sender applications.

2.1.2 HTTP Adapter

Feature details

The HTTP adapter allows HTTP requests to be received using HTTP communication.

The HTTP adapter extracts event data in XML or CSV format from the received HTTP requests and passes it to the high-speed filter.

It allows communication that is more lightweight than SOAP communication.

Usage scenario

The HTTP adapter is used if the event sender application is a Web application, or for smartphones.

Refer to Chapter 3, "Input Adapter Reference" in the *Developer's Reference* for information on setting request headers and also for samples of event sender applications.

2.1.3 Socket Adapter

Feature details

The socket adapter allows massive volumes of events to be received at high speed using a communication protocol unique to BDCEP.

The socket adapter passes the received events to high-speed filter processing.

Usage scenario

The socket adapter is used if massive volumes of events need to be processed using high throughput.

Refer to Chapter 3, "Input Adapter Reference" in the *Developer's Reference* for protocol details and also for samples of event sender applications.

2.2 Logging

Logging is a feature that records, as a log, events received from the input adapter prior to high-speed filter processing or events that are the output results of complex event processing.

An Interstage Big Data Parallel Processing Server (Hadoop collaboration) or a CEP Server can be selected as the destination for recording events.

Recording in an Interstage Big Data Parallel Processing Server (Hadoop collaboration)

This selection records events in an event log to be generated in the Interstage Big Data Parallel Processing Server (hereafter, referred to as "BDPP").

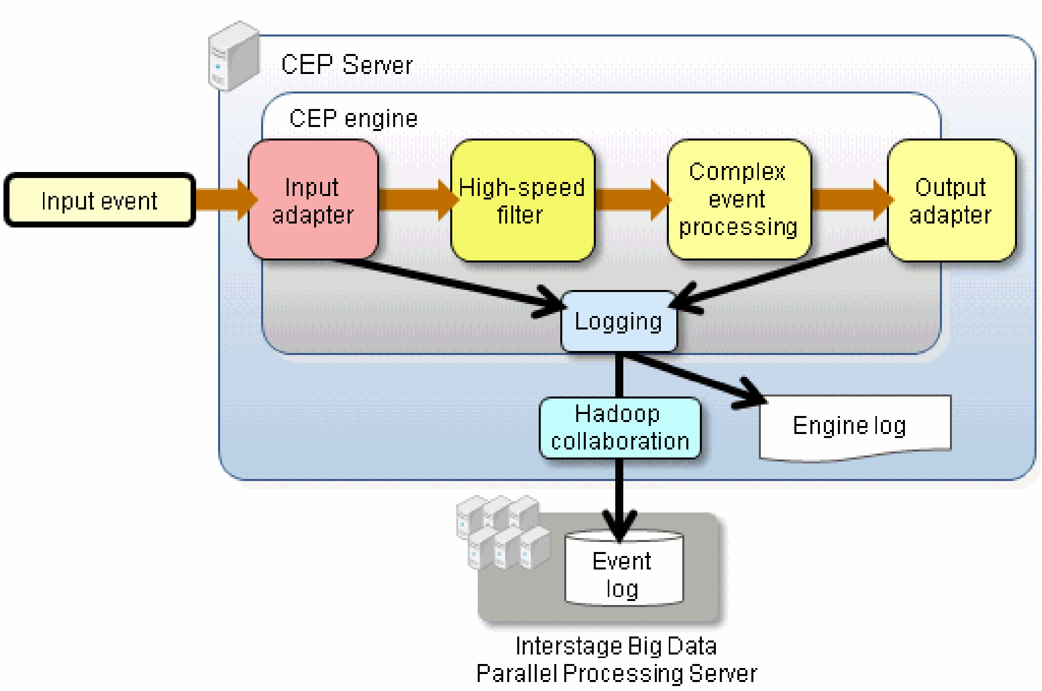
Accumulation, analysis, and processing of the event log can be performed in the BDPP.

The analysis results of the event log can be used for improving the accuracy of the complex event processing rules.

Recording in a CEP Server

This selection records events in the engine log of a CEP Server. The engine log can be used for purposes such as checking event reception and checking rule operation when rules are being developed.

Figure 2.2 Logging



Select the destination for recording events according to your objective.

Refer to "5.6.4.1 Output Destination and File Format of an Event Log" for information such as the output destination of the event log.

Note

To use Hadoop collaboration, BDPP must be installed separately.

2.3 High-speed Filter

The high-speed filter is a feature that allows the "extraction process of input events" and "join processing of input events with master data" to be performed at high speed.

Events output by the high-speed filter become input events for complex event processing.

This section explains the filter rules used by the high-speed filter as well as the master data (files in which data from the master database is stored in CSV format).

2.3.1 Filter Rules

Filter rules define how the "extraction process of events" and "join processing of events with master data" are to be performed.

Input events passed from the input adapter to the high-speed filter are processed based on the filter rules that have been defined.

Filter rules are described using IF-THEN-type format.

Refer to Chapter 2, "Filter Rule Language Reference" in the *Developer's Reference* for information on filter rules.

Typical processes that are performed using filter rules are as follows:

Extraction process of input events

Only events that match the conditions specified as filter rules are extracted from input events and passed to Complex Event Processing.

Extraction process using master data matching

The master data is matched based on the contents of input events, and input events are extracted in accordance with the matched value in the master data.

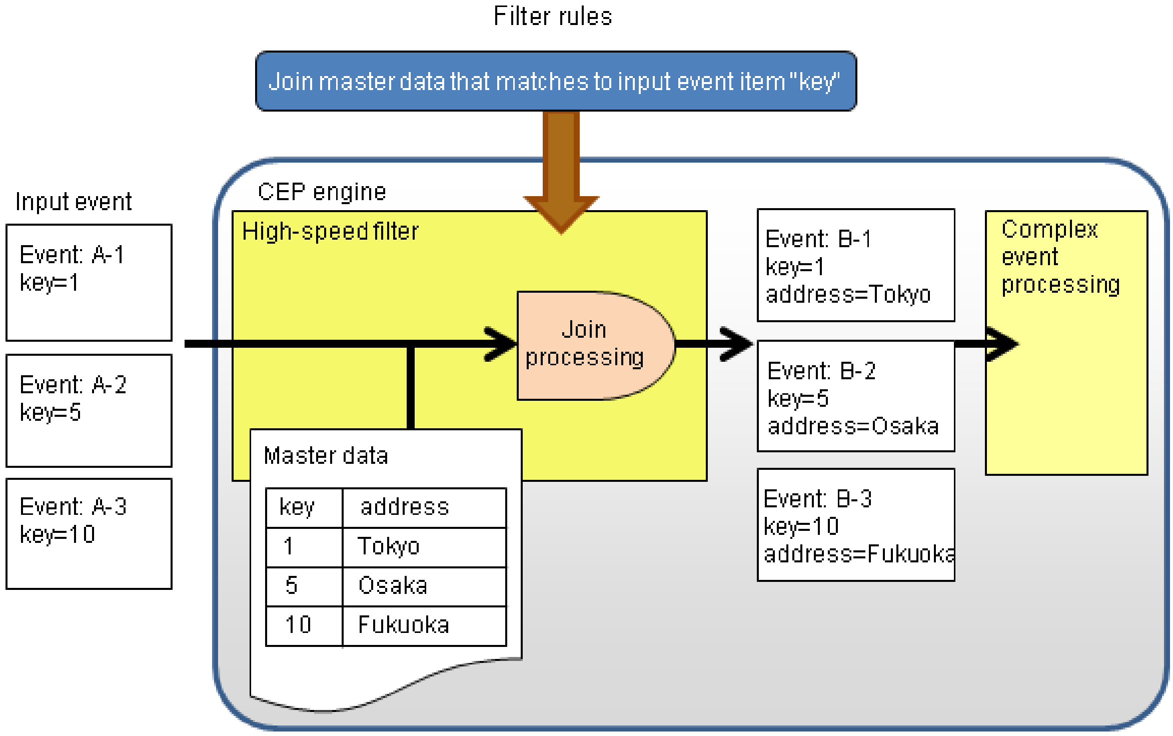
The master data is used only as conditions for the extraction process, and join processing with input events will not be performed.

Join processing with master data

Input events and pre-registered master data are joined and passed to Complex Event Processing.

The extraction process can also be performed on the join results.

Figure 2.3 Example of join processing with master data



key	address
1	Tokyo
5	Osaka
10	Fukuoka

Weighting processing of text

The text contained in input events is weighted using specific keywords that appear in the text.

This processing allows the extraction of only those events with a total weight that exceeds the threshold, and also to detect those events that are consecutively received if used with complex event processing.

2.3.2 Master Data

The master data refers to the files in which data from the master database is stored in CSV format.

The master data is used as "join targets" or "extraction conditions of input events" that have been passed to the high-speed filter.

Refer to "5.5.2 Master Data (for the High-speed Filter)" for information on the master data.

Note

The master data is loaded in CEP engine memory when the CEP engine starts. Refer to "3.3.1 Estimating Memory Usage" for information on the required memory usage.

2.4 Complex Event Processing

Complex event processing is a feature that allows any event to be detected and allows "join processing between events" by using pre-registered event processing rules.

This section explains the features in complex event processing.

2.4.1 Features of Complex Event Processing

Complex event processing receives events processed by the high-speed filter and then processes them according to complex event processing rules.

It then passes the events that fall under the rules defined in complex event processing to the output adapter.

Refer to "2.6 Output Adapter" for information on the output adapter.

Complex event processing rules are described using the SQL-type format (complex event processing language).

Refer to Chapter 1, "Complex Event Processing Language Reference" in the *Developer's Reference* for information on complex event processing language.

The items below are features of Complex Event Processing in BDCEP.

Matching between events

With matching between events, matching is performed between multiple items of event data input from the high-speed filter.

External data access

With external data access, Terracotta collaboration and RDB collaboration are used to allow external data to be referenced.

Complex event processing allows external data to be referenced just by defining rules.

Refer to "2.5.1 Terracotta Collaboration" for details.

Refer to "2.5.2 RDB Collaboration" for details.

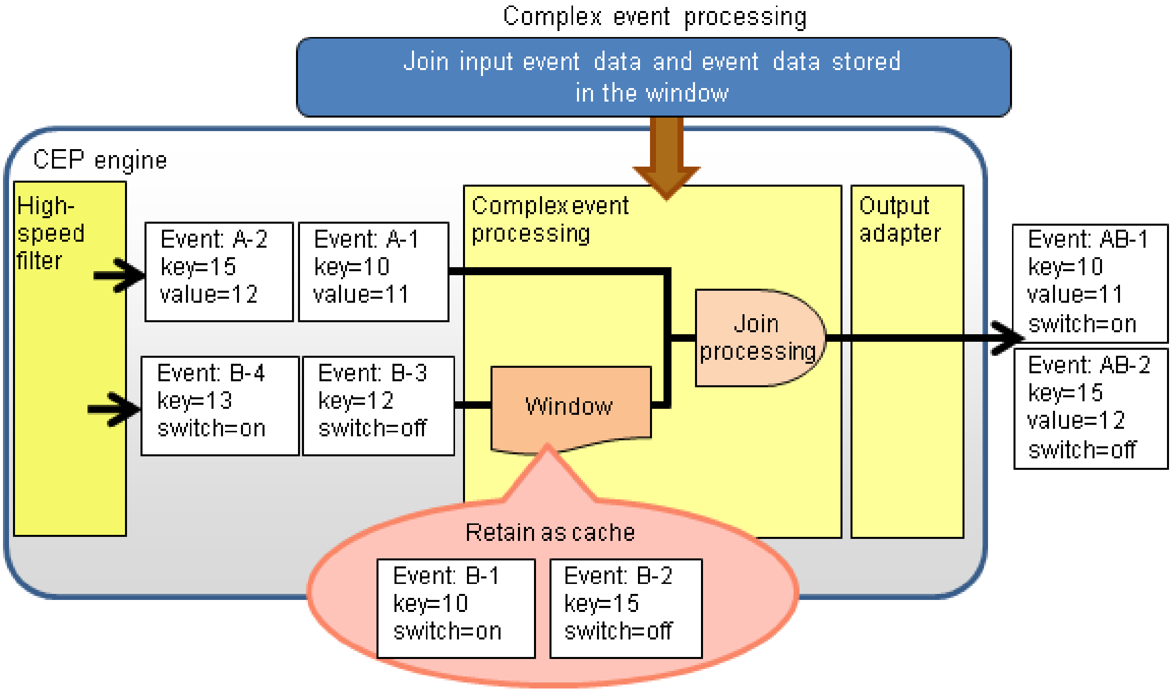
Join processing of cache data and input events

With complex event processing, input events can be retained in the memory cache (feature available in the window).

Join processing of the retained cache data with other input events is also possible.

The example below is an operation image of join processing.

Figure 2.4 Example of join processing of cache data and input events using a window


Complex event processing
Join input event data and event data stored in the window
CEP engine
High-speed filter
Event: A-2 key=15 value=12
Event: A-1 key=10 value=11
Event: B-4 key=13 switch=on
Event: B-3 key=12 switch=off
Complex event processing
Join processing
Window
Output adapter
Event: AB-1 key=10 value=11 switch=on
Event: AB-2 key=15 value=12 switch=off
Retain as cache
Event: B-1 key=10 switch=on
Event: B-2 key=15 switch=off

Note

- Memory usage must be estimated according to the cache to be retained. Refer to "3.3.1 Estimating Memory Usage" for information on the estimation method.
- If the CEP engine is stopped, the cache retained in memory will be deleted.

2.5 External Data Access

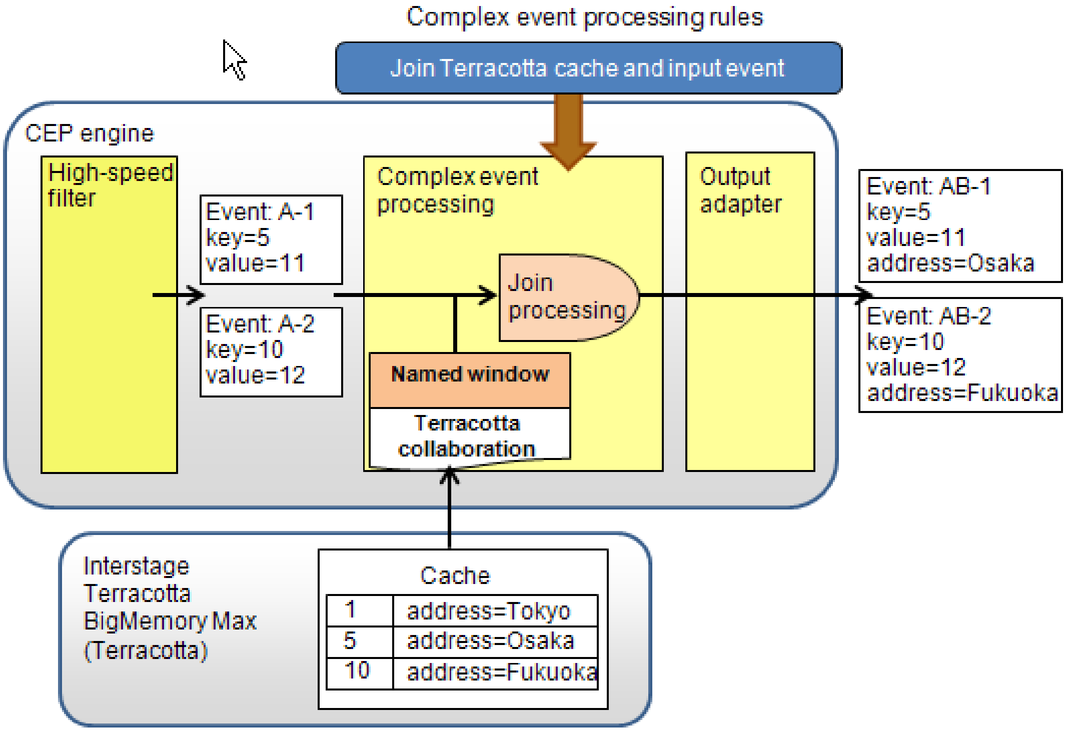
Complex Event Processing allows external data to be referenced using Terracotta collaboration or RDB collaboration. In addition, Terracotta collaboration allows data to be added, updated, and deleted.

2.5.1 Terracotta Collaboration

A cache stored in external Interstage Terracotta BigMemory Max (hereafter referred to as "Terracotta") can be referenced as a named window from the complex event processing of BDCEP. As data in the window is actually external to the system, this window is also called a Virtual Data Window.

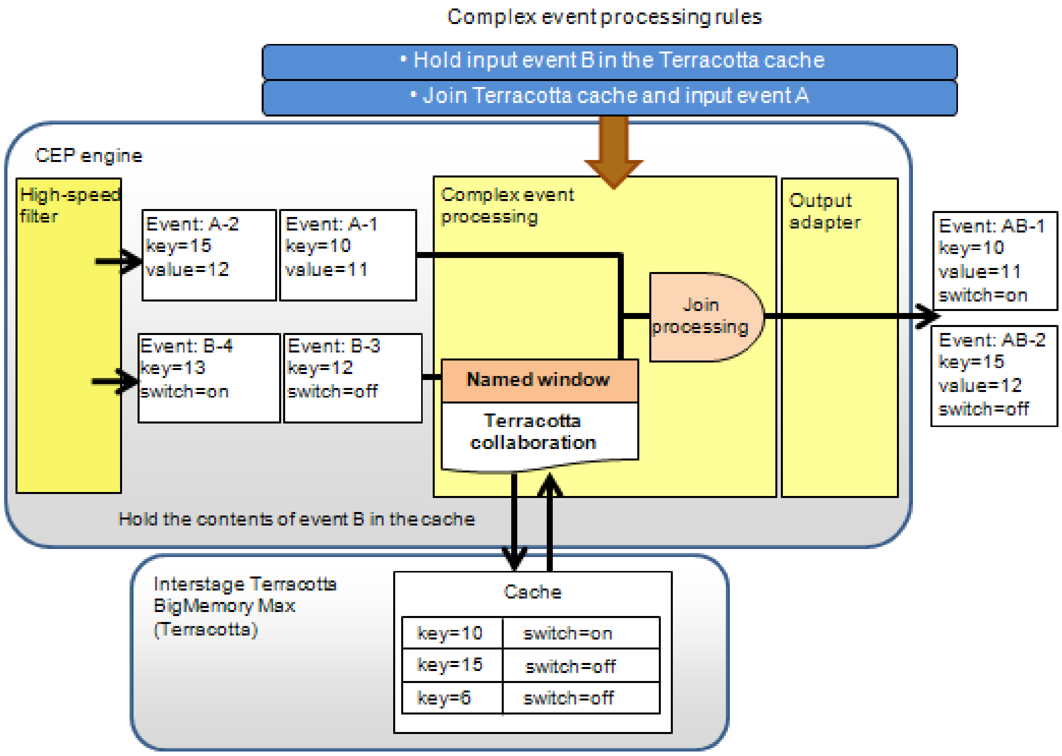
In the example below, a cache stored in Terracotta is referenced, and the referenced data is joined with input events and then passed to the output adapter.

Figure 2.5 Example of Terracotta collaboration (1)



The following example retains input events in the Terracotta cache and joins the retained cache data to other input events.

Figure 2.6 Example of Terracotta collaboration (2)



Refer to "4.4.3 Setup of Terracotta Collaboration" for information on how to set up Terracotta.

Refer to "5.4.4.3.3 Using Terracotta cache" for information on how to use a Terracotta cache from complex event processing.

Note

To use Terracotta collaboration, you must separately install Terracotta.

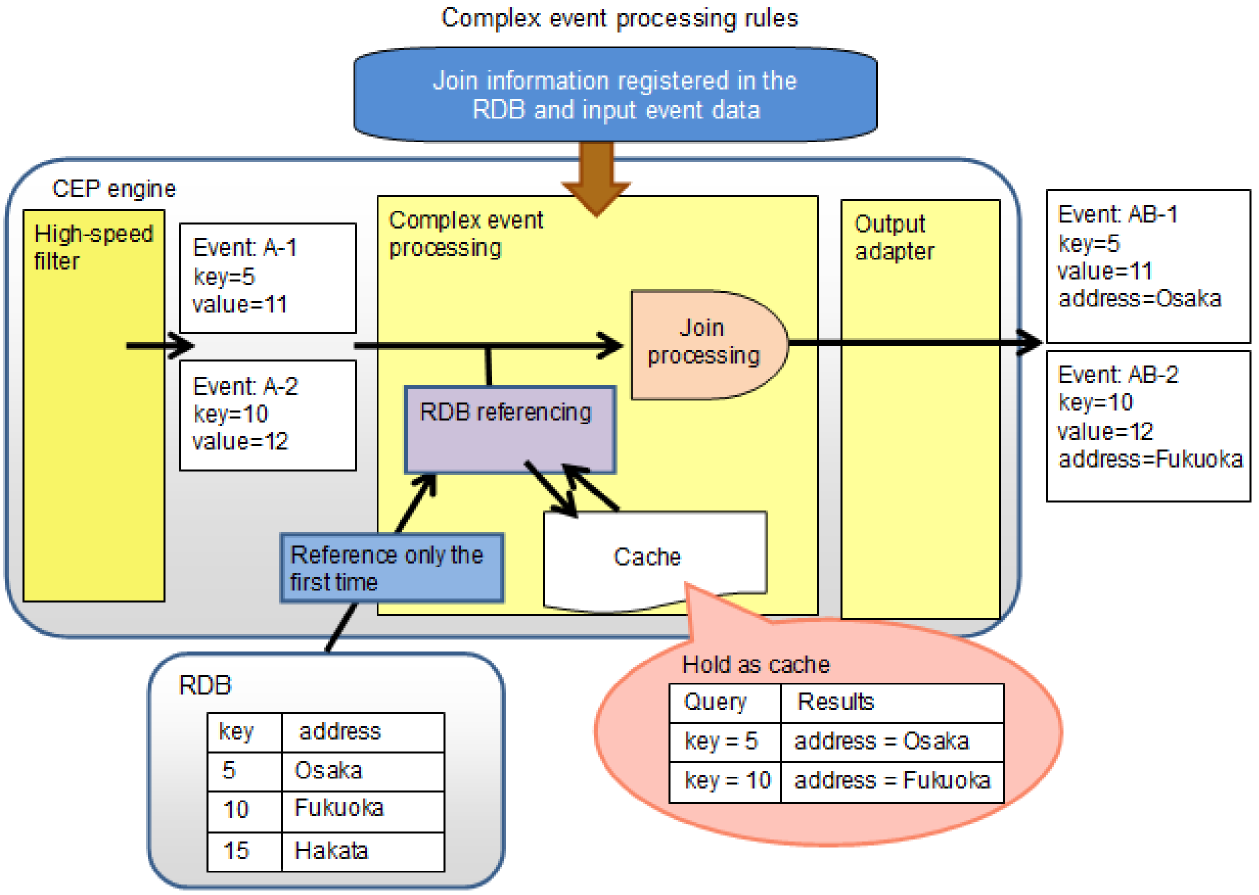
2.5.2 RDB Collaboration

You can reference RDB data from complex event processing provided by BDCEP.

Setting up the system to use a cache enables data to be referenced at a higher speed than when accessing a relational database each time. Essentially, the key and result of a query that has been referenced once are saved in the cache, and when the relational database is subsequently referenced using the same key, the data is obtained from the stored cache.

The following example references the data in a relational database, joins the referenced data to input events, and passes it to the output adapter.

Figure 2.7 Example of RDB collaboration



key	address
5	Osaka
10	Fukuoka
15	Hakata

Query	Results
key = 5	address = Osaka
key = 10	address = Fukuoka

Refer to "4.4.4 Setup of RDB Collaboration" for information on how to set up RDB collaboration.

Refer to "5.4.4.4.2 Specifying RDB referencing in complex event processing rules" for information on how to reference the data in a relational database.

Note

- If there is a large volume of data stored in the cache, memory usage also increases. Set a cache retention period according to memory usage. Refer to "3.3.1 Estimating Memory Usage" for information on how to estimate the amount of memory to be used by RDB referencing.

- RDB collaboration uses JDBC drivers provided by various RDB products. BDCEP operations are confirmed for the following JDBC drivers:
    - JDBC driver of Symfoware Server V10.1.0A
    - JDBC driver of Symfoware Server (Open Interface) V12.0
    - PostgreSQL JDBC Driver Version 9.3-1100 (JDBC 4)

2.6 Output Adapter

The output adapter is a feature that outputs the processing results of complex event processing rules externally.

BDCEP provides four output adapters shown below, according to objective of the user:

Figure 2.8 Output adapter

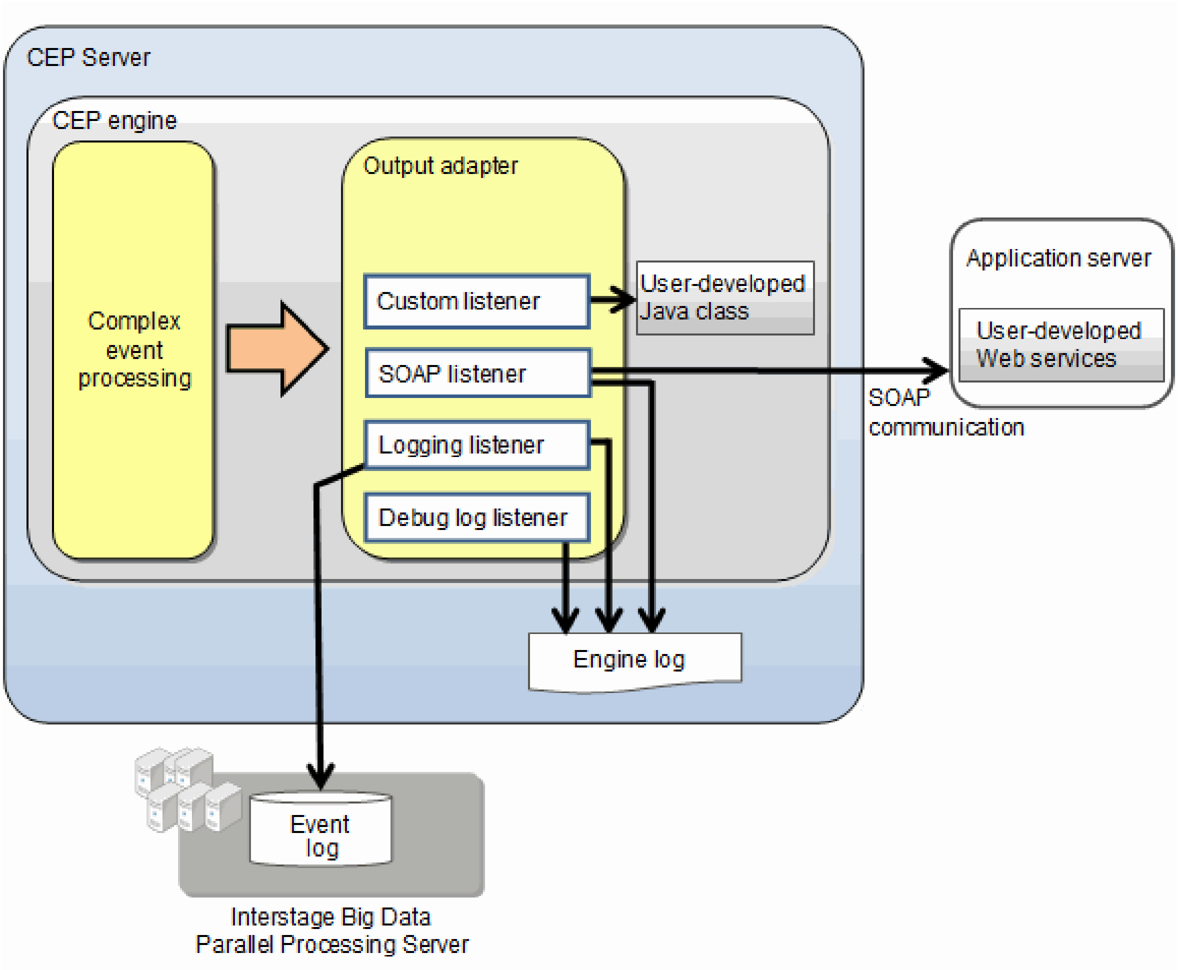


Table 2.1 Overview of the output adapter

Output adapter	Details of feature	Output destination
SOAP listener	Notifies the results of complex event processing (events) to a user-developed Web service, using SOAP communication	- User-developed Web service - Engine log (for send records)
Custom listener	Passes the results of complex event processing to a user-developed Java class, and executes them.	- User-developed Java class

Output adapter	Details of feature	Output destination
Logging listener	Logs the results of complex event processing	- Event log (BDPP) - Engine log
Debug log listener	Outputs debug information on complex event processing rules	- Engine log

2.6.1 SOAP Listener

The SOAP listener sends the results of complex event processing to a user-developed Web service, using SOAP communication.

It also leaves send records in the engine log.

Use the SOAP listener if you want to use the processing results of complex event processing in an external application.

Refer to "5.4.4.5 SOAP Listener" for information on how to use the SOAP listener.

Also refer to "5.4.8 Designing a SOAP Listener Definition".

2.6.2 Custom Listener

The custom listener passes the results of complex event processing to a user-developed Java class.

Use it to process the results of complex event processing using a Java program that runs on the same process (Java VM) as the CEP engine.

Refer to "5.4.4.6 Custom Listener" for information on how to use the custom listener.

Note

You must develop the user-developed Java class separately.

2.6.3 Logging Listener

The logging listener logs the results of complex event processing in the log storage area.

Use the logging listener if you want to analyze the results of complex event processing using a Hadoop system (BDPP).

Refer to "5.4.4.7 Logging Listener" for information on how to use the logging listener.

2.6.4 Debug Log Listener

The debug log listener outputs debug information on complex event processing rules to the engine log.

Use the debug log listener if you want to check the operation of complex event processing rules.

Refer to "5.7.1.2.1 Debug log listener" for information on how to use the debug log listener.

Note

The debug log listener may cause performance to decline, so use it only for development.

2.7 Operation Commands

The operational features of BDCEP are provided using commands.

A list of the commands provided is shown below.

Refer to "Chapter 8 Command Reference" for information on the commands.

Type	Command name	Command overview
Configuration	cepconfigeng	Creates or deletes a CEP engine
Development	cepdeployrsc	- Deploys a development asset - Dynamically changes rules (*1) and the master data
	cepgetrsc	References development assets
	cepundeployrsc	Undeploys a development asset
Operation	cepdispeng	Displays the status of a CEP engine
	cepdispserv	Displays the status of the CEP service (*2)
	cepstarteng	Starts a CEP engine
	cepstartserv	Starts the CEP service (*2)
	cepstopeng	Stops a CEP engine
	cepstopserv	Stops the CEP service (*2)
Maintenance and tuning	cepgetjvmopt	References JVM options
	cepsetjvmopt	Sets JVM options
Troubleshooting	cepcollectinfo	Collects data for investigation in batch

*1: Filter rules and complex event processing rules

*2: Service that manages the CEP engines on the CEP Server

2.7.1 Dynamically Changing Rules and Master Data

The feature for dynamically changing rules and the master data enables you to replace rules (filter rules and complex event processing rules) with new rules and reload the master data without stopping the CEP engine.

This feature reduces the time required for replacing rules and reloading the master data, because it is not necessary to stop and start the CEP engine.

Refer to "8.3 cepdeployrsc" for information on dynamic change.

2.8 Engine Log

Information on errors that have occurred in a CEP engine, such as insufficient memory and logging failures, is output to the engine log.

If some abnormality is detected in a CEP engine, this log can be analyzed in order to identify the cause.

Debug information for complex event processing rules that have been described is also output to the engine log, if the debug log listener has been set in the complex event processing rules.

The engine log is also used by logging as an event recording destination.

Refer to "6.1.5.3 Monitoring Abnormalities Using Logs" for information such as the output destination of the engine log.

2.9 Resource Log

Resource information for a CEP engine that is collected on a regular basis is output to the resource log.

Analyzing this log allows resource use conditions to be monitored and tuned.

Refer to "6.1.5.4 Checking the Resource Usage of the CEP Engine" for information such as the output items and output destination of the resource log.

2.10 Cluster Service

The Cluster Service of BDCEP can be used to build a reliable system using PRIMECLUSTER, in order to prevent a long-term suspension of business due to a hardware fault on the CEP Server.

Refer to "7.2.1 Overview of Reliable System Operations" for details.

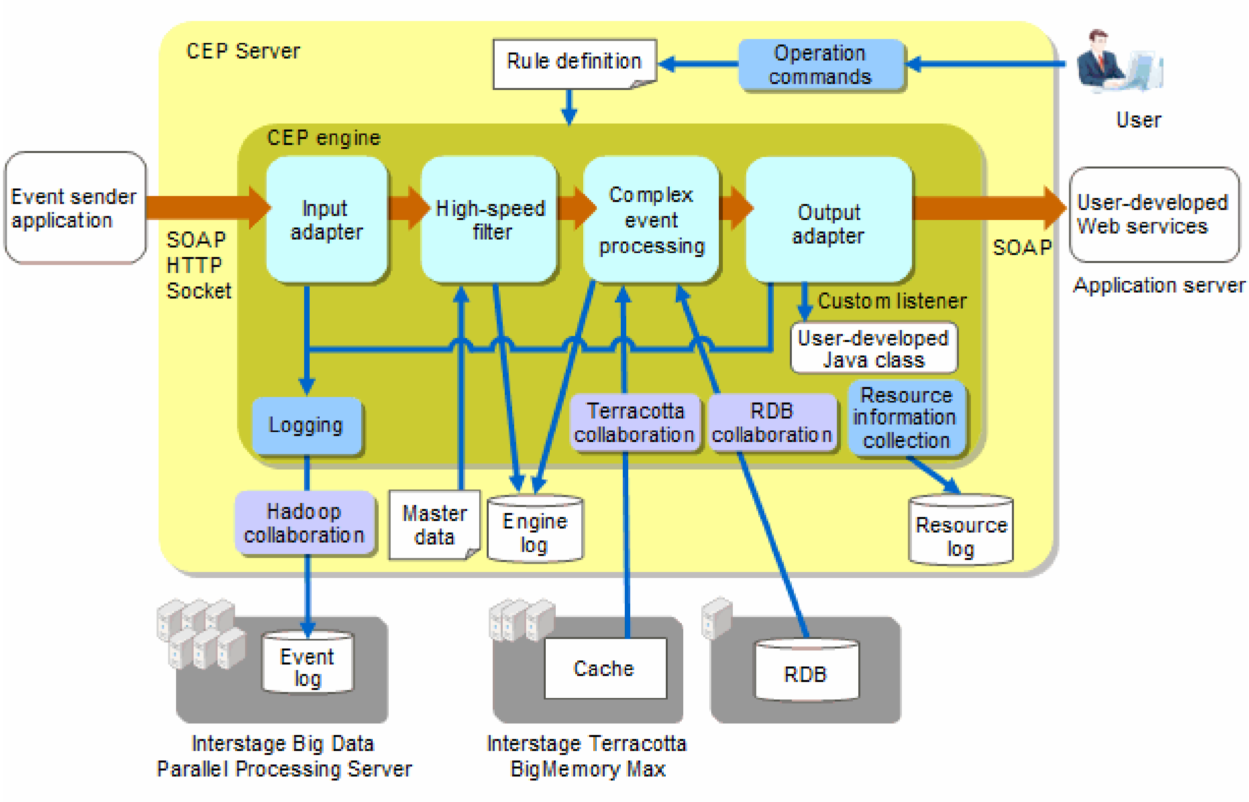
Chapter 3 System Configuration and Design

This chapter explains the system configuration and design of Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP").

3.1 System Configuration

A configuration diagram of BDCEP and related products is shown below.

Refer to "Chapter 2 Features Provided" for information on each feature.



Collaboration features

BDCEP allows collaboration with the products listed below. Note that in the given product names. *x* can be any number and *X* can be any letter.

- Product that can be used in Hadoop collaboration
    - Interstage Big Data Parallel Processing Server V1.0.*x* (V1.0.1 or later)
- Product that can be used in Terracotta collaboration
    - Interstage Terracotta BigMemory Max V4.0.*x* (V4.0.1 or later)
- Products that can be used in RDB collaboration (can collaborate with relational databases supported by the JDBC drivers of the following products)
    - Symfoware Server V10.*x*.*xX* (V10.1.0A or later)
    - Symfoware Server (Open Interface) V12.*x*.*x* (V12.0.0 or later)
    - PostgreSQL JDBC Driver Version 9.*x*-*xxxx* (JDBC 4) (Version 9.3-1100 or later)

If the collaboration features mentioned above are to be used, a server with the product installed must be provided for each product, in addition to the CEP Server.

Refer to the manual of each product for information on designing the server configuration and also for the installation procedure required for each product.

Refer to "4.4 Setup" for information on the setup of a CEP Server for collaboration.

3.2 Designing the System Configuration

This section explains designing the system configuration.

3.2.1 Designing the System Configuration

After identifying which features and collaboration systems will be required, perform the design tasks explained in the sections below according to the tasks where BDCEP is to be used and the purpose for using BDCEP.

Refer to "3.1 System Configuration" and "Chapter 2 Features Provided" for information on the system configuration and for details on each feature.

Refer to "7.1 Scalable System Operations" if operating BDCEP in a scalable configuration, or "7.2 Operating a Highly Reliable System Using PRIMECLUSTER" if operating BDCEP in a highly reliable configuration.

3.2.2 Aspects of Designing the CEP Server

After deciding on the features to be used for each business application, estimate the CEP Server configuration. To achieve a design that meets the estimates requirements use the following features:

- Overall design
- Designing the input adapter
- Designing the high-speed filter
- Designing complex event processing
- Designing the output adapter

The following sections explain the considerations required for the design of each feature. The items considered here are used in "3.3 Designing System Resources".

3.2.2.1 Overall Design

The main consideration for the overall design of the CEP Server is shown below.

- Number of CEP engines to operate on the CEP Server

3.2.2.2 Designing the Input Adapter

The input adapter design considerations are as follows:

- Number of input event type
- Input event details
    - Data received (per unit of time)
    - Maximum data size
    - Average data size
    - Use of logging
    - Number of items

3.2.2.3 Designing the High-speed Filter

The high-speed filter processing design considerations are as follows:

- Number of high-speed filter statements (IF-THEN statements)
- High-speed filter statement details (IF-THEN statement)
    - Search conditions to be specified in the high-speed filter statement
    - Master data to be used
    - High-speed filter processing event types (those where input events and item content are different)
- High-speed filter processing event details
    - Number of occurrences (per unit of time)
    - Average data size
    - Number of items
- Number of master data
- Master data details
    - Number of records
    - Average data size of each item
    - File size

3.2.2.4 Designing Complex Event Processing

The complex event processing design considerations are as follows:

- Use of Terracotta collaboration
- Number of Terracotta cache
- Use of RDB collaboration
- Details of the cache size to be used for RDB collaboration
    - Cache retention period
    - Cache purge interval

3.2.2.5 Designing the Output Adapter

The output adapter design considerations are as follows:

- Number of user-developed Web service (called from complex event processing rules)
- Details of calls to user-developed Web services
    - Number of calls (per unit of time)
    - Average data size
- Number of data type (events or processing results of rules) for complex event processing rules logging
- Details of data to be logged
    - Number of occurrences (per unit of time)
    - Average data size
- Types of user-developed Java classes (custom listener) called from complex event processing rules
- Details of calls to user-developed Java classes
    - Number of calls (per unit of time)
    - Average data size

3.2.3 Aspects of Designing a Hadoop System for Collaboration

The Hadoop system design considerations, when using Hadoop collaboration to perform logging, are listed below.

Based on this information, design storage areas for the required event logs in the Hadoop system. Refer to the Interstage Big Data Parallel Processing Server manuals for information on the design of the Hadoop system.

- Number of data type to be logged (events or processing results of complex event processing rules)
- Details of data to be logged
    - Number of occurrences (per unit of time)
    - Average data size
    - Accumulation period

3.2.4 Aspects of Designing a Terracotta Server for Collaboration

The Terracotta server design considerations, if Terracotta collaboration is to be performed, are listed below.

Based on this information, design a Terracotta server. Refer to the Interstage Terracotta BigMemory Max manual for information on the design of the Terracotta server.

- Total data size to be stored in the cache

3.2.5 Aspects of Designing an RDB Server for Collaboration

The RDB server design considerations, if RDB collaboration is to be performed, are listed below.

Based on this information, design an RDB server. Refer to the manual for the collaborating relational database product for information on the design of the RDB server.

- Number of simultaneous RDB server connections
    - Calculate using *numberOfCepEnginesToPerformRdbCollaboration* x 2
- Details of queries issued during RDB collaboration
    - Number of queries issued per unit time
    - SQL statements of a query

3.3 Designing System Resources

3.3.1 Estimating Memory Usage

The expression for calculating the amount of memory to be used by the CEP Server is shown below.

```
Estimated amount of memory required
     = A + ((B + C + D + E + F) x numberOfCepEngines + G + H) x 1.2 + I
```

Table 3.1 Explanation of items in memory estimation expression

Item	Explanation	Memory usage
A	Base memory amount	2.7 GB
B	Amount of memory when using high-speed filter rules	Refer to "3.3.1.1 Amount of Memory when Using High-speed Filter Rules".
C	Amount of memory when master data is used by the high-speed filter	Refer to "3.3.1.2 Amount of Memory when Master Data is used by the High-speed Filter".

Item	Explanation	Memory usage
D	Amount of memory when rules are used in complex event processing	The estimation expression is shown below. (MB) *numberOfRuleDefinitions* x 31 MB
E	Amount of memory when an event type definition is used in complex event processing	The estimation expression is shown below. (MB) (*numberOfInputEventTypes + numberOfTypesOfEventsAlreadyProcessedByHighSpeed Filter*) x 37 MB
F	Amount of memory when a SOAP listener definition is used in complex event processing	The estimation expression is shown below. (MB) *numberOfUserDevelopedWebServices* x 2.5 MB
G	Amount of memory required for Terracotta collaboration	The estimation expression is shown below. (MB) *sizeOfDataPoolInCache* x *numberOfCepEnginesToPerformTerracottaCollaboration* Refer to "9.3.1 Terracotta Cache Configuration File" for the value of *sizeOfDataPoolInCache.*
H	Amount of memory required for RDB collaboration (using a cache)	The estimation expression is shown below. (MB) Memory requirement of CEP engines that perform RDB collaboration = (*cacheRetentionPeriod* + *cachePurgeInterval*) x *numberOfRdbSearchesWithDifferentSearchConditionsPerSecond* x *averageNumberOfRecordsThatMatchSearchConditions* x (*totalSizeOfExtractColumn(bytes)* + 15) x 13 / 1048576 Memory requirement of RDB collaboration = *totalMemoryRequirementOfCepEnginesThatPerformRdb Collaboration*
I	Amount of memory called from the custom listener and used by a user-developed Java class	Depends on the implementation of the user-developed Java class.

3.3.1.1 Amount of Memory when Using High-speed Filter Rules

The expression for calculating the amount of memory (bytes) when high-speed filter rules are used is shown below.

```
Amount of memory when using high-speed filter rules
    = Total amount of each ifThenStatementMemoryRequirement

ifThenStatementMemoryRequirement = 272 x 1024 x 1024 + 960 x 1024 x R + 16 x L + 8 x a
```

Variable	Meaning	Unit
R	Number of output items to be described in the output expressions of high-speed filter rules	Items
L	Maximum data size of input events	Bytes
a	Area to be used in partial character, character range, and numeric range search (*1)	Bytes

*1: Use the following expression to calculate this if search is to be executed with a partial character, character range, or numeric range specification:

```
numberOfPartialChars, charRange, or numericRange x numberOfKeywordChars x 2,048 Bytes
```

Use the following expression to calculate this if search is to be executed with a combination of partial character, character range, and numeric range specifications:

```
numberOfPartialChars x charRange x numericRange x numberOfKeywordChars x 2,048 Bytes
```

- For "number of partial characters", specify the number of parts delimited by a vertical bar (|).

  For example, if the search keyword "Sm(ith|ythe|ithy)John" is specified, the number of partial characters will be 3.

- For "character range", specify how many are in the range of the ASCII character code values separated by a hyphen (-).

  For example, if the search keyword "class[A-C]" is specified, that range will be 0x41 (A), 0x42 (B), and 0x43 (C), so the character range will be 3.

- For "numeric range", specify how many are in the range of numeric 1 and numeric 2, separated by a comma (,).

  For example, if the search keyword "alcohol[9,11]%" is specified, that range will be 9, 10, and 11, so the numeric range will be 3.

3.3.1.2 Amount of Memory when Master Data is used by the High-speed Filter

If master data is to be used, the amount of memory capacity shown below will be required in addition to what would be normally required.

```
Memory usage when using master data = Total amount of each ifThenStatementMemoryRequirement

ifThenStatementMemoryRequirement
                    = outputItemMemoryRequirement + joinRelationalExpressionMemoryRequirement

outputItemMemoryRequirement = N x (B + 60)

(If a numeric-type or string-type perfect match is specified)
joinRelationalExpressionMemoryRequirement = N x (216 + A)

(If partial match of a string is specified)
joinRelationalExpressionMemoryRequirement = a x N x (2 x A - (log10N or 1, whichever is greater)) x 144
```

The meaning of each variable is shown below.

Variable	Meaning	Unit
N	Number of master data records	Records
A	Average data size of master items specified in search expressions or in join-relational expressions in join expressions (*1)	Bytes
B	Average data size of master items specified in the output items of output expressions (*1)	Bytes
a	Join key coefficient (*2)	$0 < a < 1$

*1: If the "val" function is specified in a join-relational expression and an output item, the data size will be 16.

*2: This depends on the content of the master data specified in the join conditions in the high-speed filter rules (see the table below).

Table 3.2 Join key coefficient

Content of master data	Join key coefficient
If values vary widely in the second half of the key Example: (000001, 000002, 000012, 000125, etc.)	0.4
If values vary widely in the first half of the key Example: (100-001, 210-001, 321-001, etc.)	0.6
If values vary widely throughout the key Example: (123456, 234512, 912384, etc.)	0.8

3.3.2 Estimating Disk Usage

The expression for calculating the disk usage required by the CEP Server is shown below.

```
Disk usage required = (A + B + C + D + E + F + G + H) x 1.2
```

Table 3.3 Explanation of items in disk usage estimation expression

Item	Explanation	Directory	Estimated disk usage
A	Base disk usage	The directories are as follows: /opt /etc/opt /var/opt	The estimated disk usage is as follows: /opt: 900 MB /etc/opt: 30 MB /var/opt: 60 MB
B	Event log (Before the high-speed filter is used)	Refer to "5.6.4.1 Output Destination and File Format of an Event Log".	(*1)
C	Event log (After complex event processing)	Refer to "5.6.4.1 Output Destination and File Format of an Event Log".	(*2)
D	Resource log	High-speed filter: /var/opt/FJSVcep/cep/flt/logs/ResourceLog/*cepEngineName* Complex event processing: /var/opt/FJSVcep/cep/cep/logs/ResourceLog/*cepEngineName*	The estimation expression is shown below. (MB) 2 MB x *numberOfCepEngines*
E	Engine log	High-speed filter: /var/opt/FJSVcep/cep/flt/logs/EngineLog/*cepEngineName* Complex event processing: /var/opt/FJSVcep/cep/cep/logs/EngineLog/*cepEngineName*	The estimation expression is shown below. (MB) 200 MB x *numberOfCepEngines*
F	Custom log	/var/opt/FJSVcep/cep/cep/logs/EngineLog/*cepEngineName*	The estimation expression is shown below. (MB) 100 MB x *numberOfCepEnginesToBeUsed*
G	Master data	This will be the path specified in "dataFile" in the master definition file. Refer to "9.2.3 Master Definition File" for details.	This will be the master data file size.
H	Maintenance log of the high-speed filter	/var/opt/FJSVisjee/nodeagents/ijna/*cepEngineName*_Flt_Ins/current	The estimation expression is shown below. (MB) (*totalNumberOfFilterStatementsDescribedInHighSpeedFilterRules* x 1.6 + 0.2) x 4 MB
I	Other maintenance logs (except for the above logs)	Logs in the following directories: /var/opt/FJSVisjee /var/opt/FJSVjs2su /var/opt/FJSVcep /var/opt/FJSVihs /var/opt/FJSVjs5	The estimation expression is shown below. (MB) 1301 MB + 618 MB x *numberOfCepEngines*

Item	Explanation	Directory	Estimated disk usage
		/var/opt/FJSVj2ee /var/opt/FJSVisjmx /var/opt/FJSVisas /var/opt/FJSVod	
J	File output by the user-developed Java class (excluding the custom log)	Depends on the implementation of the user-developed Java class.	Disk usage depends on the implementation of the user-developed Java class.

*1: Calculate the events to be logged by the input adapter. The expression is shown below.

```
Disk usage of event log (B) (KB) = Total amount of each diskUsageOfEventsToBeLogged

diskUsageOfEventsToBeLogged (KB)
 = numberOfEventsReceivedPerSecond x averageDataSize (KB) x eventLogAccumulationPeriod (Seconds) x 1.2
```

*2: Calculate the data (events or processing results of rules) to be logged by complex event processing. The expression is shown below.

```
Disk usage of event log (C) (KB) = Total amount of each diskUsageOfDataToBeLogged

diskUsageOfDataToBeLogged (KB)
= numberOfDataOccurrencesPerSecond x averageDataSize (KB) x eventLogAccumulationPeriod (Seconds) x 1.2
```

Chapter 4 Installation and Setup

This chapter explains how to install, set up, and uninstall Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP").

Refer to "7.2.3 Building a Cluster Service Environment" before operating BDCEP in a high-reliability (failover cluster) environment.

4.1 Installation Overview

This section provides an overview of BDCEP installation.

4.1.1 Installation Methods

BDCEP is installed using shell scripts. The following installation methods are available:

Attended installation

Use 'attended installation' to execute the installation according to your specific requirements such as the engine execution user name.

Unattended installation

Use 'unattended installation' to execute the installation according to a setup file specified when the Installer starts, with no querying from the Installer.

4.1.2 Installed Packages

Below is a list of the packages installed by BDCEP.

Table 4.1 List of packages

Type	Package name	Note
Basic features of BDCEP	FJSVcep	
	FJSVes	
	FJSVextp	
	FJSVihs	
	FJSVisas	
	FJSVisco	
	FJSVisgui	
	FJSVisjee	
	FJSVisjmx	
	FJSVisscs	
	FJSVj2ee	
	FJSVJavaSE-jdk6-rhel5	Installed on Red Hat Enterprise Linux 5 (for Intel64)
	FJSVJavaSE-jdk6-rhel6	Installed on Red Hat Enterprise Linux 6 (for Intel64)
	FJSVjs2su	
	FJSVjs5	
	FJSVjssrc	
	FJSVod	
	FJSVots-EE	

Type	Package name	Note
	FJSVporb	
	FJSVsclr64	
	FJSVsmee64	
	FJSVsvmon	
	FJSVtd	
	FJSVtdis	
	FJSVxmlpc	

Note

If packages provided by BDCEP are installed or uninstalled directly, for example by using rpm, they will not operate normally.

Unless directed to do otherwise by Fujitsu technical support, always use the BDCEP installation and uninstallation shell scripts to install and uninstall, respectively.

4.2 Installation Requirements

This section explains the resources required for installation.

4.2.1 Hardware Environment

The hardware below is required in order to use BDCEP:

- PRIMERGY RX series or PRIMERGY TX series

BDCEP also requires an environment with sufficient memory available.

Refer to "Chapter 3 System Configuration and Design" for information on estimating the memory size.

4.2.2 Software Environment

The software below is required in order to use BDCEP.

4.2.2.1 Required Operating System

Either of the operating systems below is required.

Operating system name	Remarks
Red Hat Enterprise Linux 5 (for Intel64)	This operating system supports operation using version 5.3 or later.
Red Hat Enterprise Linux 6 (for Intel64)	This operating system supports operation using RHSA-2010:0842 (kernel-2.6.32-71.7.1.el6) or later. Version 6.1 or later has RHSA-2010:0842 applied.

Note

BDCEP is guaranteed to operate in an environment where the SELinux function is disabled.

Information

If BDCEP is to operate on Red Hat Enterprise Linux 6 (for Intel64), use the packages below in addition to the packages installed with the minimum operating system option.

Package	Architecture
alsa-lib	x86_64
cloog-ppl	x86_64
compat-libtermcap	i686
compat-readline5	i686
cpp	x86_64
file	x86_64
gcc	x86_64
gcc-c++	x86_64
gdb	x86_64
glibc	i686
glibc-devel	x86_64
glibc-headers	x86_64
kernel-headers	x86_64
libICE	x86_64
libSM	x86_64
libX11	x86_64
libX11-common	noarch
libXau	x86_64
libXext	x86_64
libXi	x86_64
libXp	x86_64
libXt	x86_64
libXtst	x86_64
libgomp	x86_64
libstdc++-devel	x86_64
libtool-ltdl	x86_64
libxcb	x86_64
lksctp-tools	x86_64
make	x86_64
mpfr	x86_64
ncurses-libs	i686
nss-softokn-freebl	i686
perl	x86_64
perl-Module-Pluggable	x86_64
perl-Pod-Escapes	x86_64
perl-Pod-Simple	x86_64

Package	Architecture
perl-libs	x86_64
perl-version	x86_64
ppl	x86_64
redhat-lsb	x86_64
strace	x86_64
tcsh	x86_64
unixODBC	x86_64
zlib	i686

4.2.2.2 Mandatory Patch

The patch below must be installed in advance.

Item No.	Operating system	Patch ID and batch update	Remarks
1	Red Hat Enterprise Linux 6 (for Intel64)	RHBA-2011:0321-1	

4.2.2.3 Required Packages

The packages below are required for using BDCEP.

If you install BDCEP in an environment where the following packages have not been installed, the BDCEP installer installs them.

Item No.	Package name	Remarks
1	FJSVcir (CIRuntime Application)	This is "Uninstall (middleware)", a common tool for Fujitsu middleware products. It manages information about installed Fujitsu middleware products and deletes products.
2	FJSVqstl (FJQSS)	This is a data collection tool common to Fujitsu middleware products.

4.2.2.4 Mutually Exclusive Software

Do not install the software or packages below on the same system as BDCEP.

Item No.	Product name	Version	Remarks
1	Interstage Application Server	V9.0.0 or later	(*1)
2	Interstage Application Development Cycle Manager	V10.1 or later	
3	Interstage Big Data Complex Event Processing Server	V1.0.0 or later	No more than one instance can be installed on the same operating system.
4	Interstage Big Data Parallel Processing Server	V1.0.0 or later	The Development Server can be installed on the same system.
5	Interstage Business Application Server	8.0.0 or later	(*1)
6	Interstage Service Integrator	V9 or later	(*1)
7	Interstage Job Workload Server	V8 or later	
8	Interstage List Works	V9 or later	
9	Interstage Service Integrator	V9.0.0 or later	(*1)

Item No.	Product name	Version	Remarks
10	Interstage Shunsaku Data Manager	V7	
11	Interstage Web Server	V9.0.0 or later	(*1)
12	Interstage Web Server Express	V11.0.0 or later	(*1)
13	ServerView Resource Orchestrator Cloud Edition	V3 or later	
14	Systemwalker Availability View	V13.3.0 or later	(*1)
15	Systemwalker Centric Manager	V13.4.0 or later	The Job Server can be installed on the same system.
16	Systemwalker IT Change Manager	V14 or later	
17	Systemwalker Network Manager	V12 or later	
18	Systemwalker Service Catalog Manager	V14.1 or later	
19	Systemwalker Service Quality Coordinator Enterprise Edition	V13.4 or later	This is mutually exclusive software only when a dashboard or BrowserAgent is used.
20	Systemwalker Software Configuration Manager	V14.1 or later	

*1: In Red Hat Enterprise Linux 5 (for Intel64) or Red Hat Enterprise Linux 6 (for Intel64), the product cannot be installed on the same system even if the product supports operation in 32-bit mode.

4.2.3 Resources Required at Installation

The resources below are required for BDCEP installation.

Disk capacity

The disk capacity below is required for installing BDCEP. If necessary, extend the size of the relevant file system.

Directory	Required disk capacity
/opt	900 MB
/etc/opt	30 MB
/var/opt	60 MB

Memory capacity

The memory capacity below is required for installing BDCEP. If necessary, extend the amount of memory installed.

- 2.7GB

4.2.4 Resources Required at Operation

To operate BDCEP, estimate the following resources and allocate the required capacities:

- Memory
- Disk

Refer to "Chapter 3 System Configuration and Design" for information on estimating resources.

4.3 Installation

This section explains how to install BDCEP.

4.3.1 Pre-installation Procedure

This section explains the tasks required before installing BDCEP.

- Setting /etc/hosts
- Checking the Port Numbers to be Used
- Checking Free Disk Capacity
- Creating the Engine Execution User and Group
- Checking Kernel Parameters
- Checking Resource Limitations
- Modifying /etc/cron.daily/tmpwatch in Red Hat Enterprise Linux 5.4 or Earlier Version
- Deleting FJSVsmee64 and FJSVsclr64 Packages

4.3.1.1 Setting /etc/hosts

Configure the /etc/hosts file so that network name resolution is enabled for the CEP Server host name (*1). Note the following when registering the host name:

- When registering the CEP Server host name as "127.0.0.1" (CEP Server loopback address), always first describe the IP address setting used when gaining access from outside the CEP Server.
- Alternatively, do not set the host name of the CEP Server for "127.0.0.1".

*1: The HOSTNAME parameter setting in the /etc/sysconfig/network file.

Example

Example of setting /etc/hosts

Below is an example of setting the host name "cepsv1" for "127.0.0.1". In this example, the IP address setting used when gaining access from outside the CEP Server is "10.10.10.10".

```
10.10.10.10  cepsv1
127.0.0.1    cepsv1    localhost.localdomain  localhost
```

4.3.1.2 Checking the Port Numbers to be Used

Check that the port numbers to be used by BDCEP are available for use. The port numbers used by BDCEP are shown below. Use firewall or operating system settings to ensure the relevant port can be used.

Port number	Description
80	Port number used by the input adapter (SOAP adapter or HTTP adapter) to receive input events.
81	Port number used internally by BDCEP. Access from outside the CEP Server is not required.
102	
389	
636	
2000	
2465	
3279	
4433	
5432	

Port number	Description
6666	
8002	
8009	
8080	
8686	
8909	
8919	
9700	
10550	
10555	
12000	
12001	
12200	
12210	
12220	
12230	
13000	
23600-23602	
23700-23710	
28080	
28090-28100	
28686-28696	
anyPort	Port number used by the input adapter known as the socket adapter to receive input events. Use this by setting one unused port number between 5001 and 32767 for each CEP engine, and by setting a maximum of five for the entire CEP Server. Setting the port numbers to be used is done in CEP engine setup after installation. Port number 9600 is set in the CEP engine created at initial setup.

4.3.1.3 Checking Free Disk Capacity

Check that the disks have sufficient free capacity. Refer to "4.2.3 Resources Required at Installation" for information on the required disk capacity.

If there is a shortage of free disk capacity, extend the size.

4.3.1.4 Creating the Engine Execution User and Group

Create a specific user and group to execute the CEP engine. Each of the processes of the CEP engine run with the user and group permissions created here.

Example

Operation example of creating the engine execution user and group using the user name "isbdcep" and the group name "isbdcep"

```
$ su - <ENTER>
# /usr/sbin/groupadd isbdcep<ENTER>
# /usr/sbin/useradd -g isbdcep isbdcep<ENTER>
```

Note

- The user creation method depends upon the management policy of the system. Always check with the system administrator.
- Up to 8 characters can be specified for the user name and the group name.

4.3.1.5 Checking Kernel Parameters

Kernel parameters must be tuned in advance when operating BDCEP.

Edit `/etc/sysctl.conf` to change the values of the target kernel parameters to suitable values, according to the parameter "type".

- If the type is "Maximum":

  If the value already set (initial value or previously set value) is greater than the value shown in the table, it need not be changed. If it is smaller than the value in the table, change it to the value in the table.

- If the type is "Additional"

  Add the value shown in the table to the value that is already set (initial value or previously set value). Check the system maximum values before adding this value and, if adding that value would exceed the system maximum value, set the system maximum value.

The current kernel parameters can be verified using "`/sbin/sysctl -a`".

After making the changes, execute "`/sbin/sysctl -p /etc/sysctl.conf`" or reboot the OS.

Refer to the documentation for the operating system for information on how to change the kernel parameters.

Below are the kernel parameters to be set.

Shared memory

Parameter	Description	Value to be set	Type
kernel.shmmax	Maximum size in shared memory	57413492	Maximum
kernel.shmmni	Maximum number of shared memory segments	41	Additional

Semaphore

For semaphore settings, set the values for each parameter in the following format:

```
kernel.sem = SEMMSL SEMMNS SEMOPM SEMMNI
```

Parameter	Description	Value to be set	Type
SEMMSL	Maximum number of semaphores for each semaphore identifier	512	Maximum
SEMMNS	Number of semaphores for the system as a whole	5763	Additional
SEMOPM	Maximum number of operators for each semaphore call	50	Maximum
SEMMNI	Number of semaphore operators for the system as a whole	1143	Additional

Message queue

Parameter	Description	Value to be set	Type
kernel.msgmax	Maximum message size	16384	Maximum
kernel.msgmnb	Maximum number of bytes of messages in the message queue	32768	Maximum
kernel.msgmni	Maximum number of message queue IDs	526	Additional

4.3.1.6 Checking Resource Limitations

When operating BDCEP, adjust the user limitations for the number of processes (threads) that can be executed.

Edit "`/etc/security/limits.conf`" and change the number of processes (threads) that the user can execute to an appropriate value.

For BDCEP, set the number of processes (threads) the engine execution user can execute to "2048" or higher. Add "2048" to the already specified value (the initial or previously specified value).

The number of processes (threads) that the engine execution user can execute can be checked with the following command as a superuser.

```
# /bin/su -c 'ulimit -u' engineExecutionUser <ENTER>
```

Reboot the OS after changing the value.

Refer to the OS manual for information on how to change the value.

Example

Example for setting /etc/security/limits.conf

This is an example for specifying the number of processes (threads) an engine execution user can execute. In this example, the value is set by adding "2048" to the default value of "1024".

```
isbdcep          soft    nproc           3072
```

4.3.1.7 Modifying /etc/cron.daily/tmpwatch in Red Hat Enterprise Linux 5.4 or an Earlier Version

The following shared memory file is generated while BDCEP is running (*XXX* is the user name, *YYY* is the process ID):

```
/tmp/hsperfdata_XXX/YYY
```

If the tmpwatch shell script is registered in cron, the shared memory file is deleted by the tmpwatch shell script. This may cause problems during product operations.

To avoid this, modify the /etc/cron.daily/tmpwatch shell script so that shared memory files are not the targets of deletion by tmpwatch.

Example

Example of modified /etc/cron.daily/tmpwatch shell script

Precede the '/usr/sbin/tmpwatch *XXX* /tmp' line with the lines in the example below (*XXX* is in hours):

```
for f in `echo /tmp/hsperfdata_*/*` ; do
  /bin/touch $f > /dev/null 2>&1
done
```

4.3.1.8 Deleting FJSVsmee64 and FJSVsclr64 Packages

If the FJSVsmee64 and FJSVsclr64 packages were installed using a Fujitsu product other than BDCEP (such as Systemwalker Centric Manager), perform the following procedure:

1. Check the installation of the FJSVsmee64 and FJSVsclr64 packages.

   Check if the FJSVsmee64 and FJSVsclr64 packages have been installed. If so, check the version of each package using the following commands:

   ```
   # rpm -q -i FJSVsmee64 | grep Version <ENTER>
   # rpm -q -i FJSVsclr64 | grep Version <ENTER>
   ```

   If the packages have been installed, the version information will be displayed. If nothing is displayed, the packages have not been installed, so the following steps are not required. Skip to "4.3.2 Installation Procedure".

   Note

   If the installed version is more recent than the version provided by BDCEP, you must perform the procedure explained in "4.3.3.2 Reinstalling FJSVsmee64 and FJSVsclr64 Packages" after installing BDCEP.

   Information

   The versions of FJSVsmee64 and FJSVsclr64 bundled with this version of BDCEP are as follows:

   ```
   FJSVsmee64  4.1.2
   FJSVsclr64  2.0.7
   ```

2. Stop all Fujitsu products. Refer to the manual provided with each product for information on how to stop the product.

3. Uninstall the FJSVsmee64 and FJSVsclr64 packages.

   ```
   # rpm -e FJSVsmee64 <ENTER>
   # rpm -e FJSVsclr64 <ENTER>
   ```

4.3.2 Installation Procedure

From the following two types, select the most suitable installation method. If installing using multi-user mode, check that the operations of other users will not affect the installation:

- Attended installation

- Unattended installation

Note

BDCEP cannot be installed in an environment where it has already been installed.

4.3.2.1 Attended Installation

1. Log in as a superuser.

   ```
   $ su - <ENTER>
   ```

2. Load the installation DVD-ROM and execute the installation shell script (install.sh) stored on the DVD-ROM from any directory.

   ```
   # mount /dev/deviceFileName dvdRomMountDir <ENTER>
   # dvdRomMountDir/install.sh <ENTER>
   ```

3. The product name will be displayed as shown below.

   ```
   +------------------------------------------------------------+
   | Interstage Big Data Complex Event Processing Server V1.1.0 |
   |                                                            |
   ```

```
|                        Copyright 2012-2013 FUJITSU LIMITED  |
+-------------------------------------------------------------+
```

4. In the interactive process shown below, which continues from the display above, specify the engine execution user and the name of the group to which the engine execution user belongs, which were created in advance.

   Only a user or group that has already been created can be specified. You can stop installation by typing "q" and pressing the Enter key.

   Below is an input example where "isbdcep" is specified as the user name and "isbdcep" is specified as the group name.

```
Please specify the engine execution user and group.

CEP engine processes will run as the specified user and group.
It differs who performs operational commands.

Please enter the user name of the engine execution user [q]: isbdcep<ENTER>

Please enter the group name of the engine execution user [q]: isbdcep<ENTER>

```

5. Then specify the engine name to be created at initial setup. You can stop installation by typing "q" and pressing the Enter key.

```
Please enter the initial engine's name (default: CepEngine) [q]: CepEngine<ENTER>
```

6. The installation details will be displayed.
   In *installPackages*, all of the packages to be installed will be displayed delimited by spaces.

```
Engine execution user (group):
  specifiedEngineExecutionUserName (specifiedEngineExecutionUserGroupName)
Initial engine's name:
  specifiedEngineName
Installation packages:
  installPackages
```

7. Check the content and, to start installing the packages, type "y" and press the Enter key.
   To stop installation, type "q" and press the Enter key.

```
Do you want to proceed with the installation? [y,q]:y<ENTER>
```

8. When installation has completed normally, the message below will be output.

```
The installation processing completed successfully.
```

9. After installation has completed normally, reboot the OS. Perform the post-installation procedure to continue.

```
# shutdown -r now <ENTER>
```

4.3.2.2 Unattended Installation

1. Create an installation file. Create the file after designing and checking the parameters to be specified in advance.
   Refer to "9.5.1 Installation File" for information on the file.

   Example

   Below is an example of the file.

   In this example, "`isbdcep`" is specified as the engine execution user, "`isbdcep`" is specified as the group to which the engine execution user belongs, and "`CepEngine`" is specified as the name of the CEP engine to be created at initial setup.

```
BDCEP_USER_NAME=isbdcep
BDCEP_GROUP_NAME=isbdcep
BDCEP_INITIAL_ENGINE_NAME=CepEngine
```

Point

A sample of the installation file is stored in "`/samples/bdcep.conf`" on the DVD-ROM of BDCEP. To work efficiently, copy the sample to the work directory of the system and then edit the parameters.

2. Log in as a superuser.

```
$ su - <ENTER>
```

3. Load the installation DVD-ROM and, from any directory, execute the installation shell script (install.sh) stored on the DVD-ROM, with the "`-s` *`installFilePath`*"option added.
   After execution, BDCEP installation will start.

```
# mount /dev/deviceFileName dvdRomMountDir <ENTER>
# dvdRomMountDir/install.sh -s installFilePath <ENTER>
```

Note

The *`installFilePath`* specification is mandatory.

If there is no file in the specified *`installFilePath`* or if the read fails, an error message will be displayed and installation will stop.

4. When installation has completed normally, the message below will be output.

```
The installation processing completed successfully.
```

5. After installation has completed normally, reboot the OS.

```
# shutdown -r now <ENTER>
```

4.3.3 Post-installation Procedure

This section explains the tasks after installation.

4.3.3.1 Setting Environment Variables

Add the path below to the `PATH` environment variable of the CEP Server users.

```
/opt/FJSVcep/bin
```

Point

Creating a file with the content below in the `/etc/profile.d` directory of the CEP Server will allow the `PATH` environment variable to be set uniformly for the CEP Server users.

File: `/etc/profile.d/FJSVcep.sh`

```
# Interstage Big Data Complex Event Processing Server V1.1.0
export PATH=/opt/FJSVcep/bin:${PATH}
```

File: `/etc/profile.d/FJSVcep.csh`

```
# Interstage Big Data Complex Event Processing Server V1.1.0
setenv PATH /opt/FJSVcep/bin:${PATH}
```

4.3.3.2 Reinstalling FJSVsmee64 and FJSVsclr64 Packages

If the versions of the FJSVsmee64 and FJSVsclr64 packages installed by BDCEP are older than the previously installed versions, perform the following procedure:

1. Uninstall the FJSVsmee64 and FJSVsclr64 packages.

   ```
   # rpm -e FJSVsmee64 <ENTER>
   # rpm -e FJSVsclr64 <ENTER>
   ```

2. Reinstall the FJSVsmee64 and FJSVsclr64 packages from a product that bundles the more recent versions of the FJSVsmee64 and FJSVsclr64 packages. Refer to the manual provided with the product for information on how to install the packages.

3. Start all products that you stopped when performing the procedure in "4.3.1.8 Deleting FJSVsmee64 and FJSVsclr64 Packages". Refer to the manual provided with each product for information on how to start the product.

4.3.3.3 Applying Updates

Refer to "6.3.3 Applying Updates" to apply product and software (included with BDCEP) updates.

After applying the updates, set up BDCEP. Refer to "4.4 Setup" for details.

4.3.4 If an Error Occurs during Installation

This section explains how to respond if a failure occurs during installation.

If an error occurs before package installation

After responding in accordance with the error message, re-execute the install.sh shell.

If an error occurs during package installation

Execute uninstall.sh and uninstall the packages that have been installed. Then, after responding in accordance with the error message output at install.sh execution, re-execute the install.sh shell.

Refer to Section 2.1, "Errors during Installation" in *Troubleshooting* for details.

Information

Installer log file

Below is the log file output by the Installer. It records detailed operation statuses in addition to the messages displayed in the windows and can be referred to when an issue arises.

```
/var/tmp/bdcep_install.log
```

When installation is successful, the log file will be stored as follows:

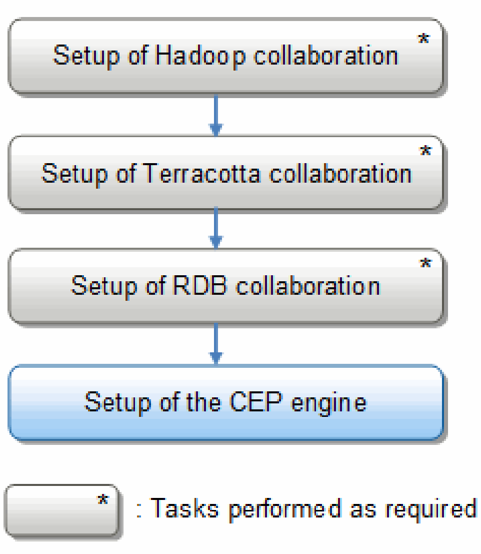
```
/var/opt/FJSVcep/bdcep_install.log
```

4.4 Setup

This section explains how to set up BDCEP.

4.4.1 Setup Overview

An overview of BDCEP setup is shown below.


Setup of Hadoop collaboration *
Setup of Terracotta collaboration *
Setup of RDB collaboration *
Setup of the CEP engine
* : Tasks performed as required

4.4.2 Setup of Hadoop Collaboration

With BDCEP, received events and events output from the CEP engine can be logged. The following two destinations can be selected for logging:

- Hadoop system
- CEP Server (*1)

*1: Events are logged in the engine log for the purpose of verifying events received and verifying the results of complex event processing.

To log events in a Hadoop system, setup of Hadoop collaboration is also required.

The Hadoop system that can be linked to is shown below. *x* can be any number.

- Interstage Big Data Parallel Processing Server V1.0.*x* (V1.0.1 or later)

Setup procedure

1. On the CEP Server, install and setup the BDPP Development Server. The CEP engine operates as an application for accessing the Hadoop system.

   Refer to Section 6.4, "Installing to a Development Server" in the *User's Guide* of the BDPP manuals for details.

2. On the Hadoop system, register the engine execution user specified at BDCEP installation as a Hadoop user.

   Refer to Chapter 11, "Managing Job Execution Users" in the *User's Guide* of the BDPP manuals for details.

4.4.3 Setup of Terracotta Collaboration

Complex Event Processing of BDCEP can collaborate with Interstage Terracotta BigMemory Max (hereafter referred to as "Terracotta") to rapidly reference the cache on Terracotta. To perform Terracotta collaboration, setup of Terracotta collaboration is also required.

The Terracotta product that can be linked to is shown below. *x* can be any number.

- Interstage Terracotta BigMemory Max V4.0.*x* (V4.0.1 or later)

Setup procedure

1. Install and set up Terracotta on the CEP Server. The CEP engine operates as a Terracotta client. Refer to the Terracotta manual for details.

2. Stop the CEP service.

```
# cepstopserv <ENTER>
```

3. Save the following four files with modified names:

   - /opt/FJSVisjee/lib/jersey-bundle-1.0.3.1.jar
   - /opt/FJSVisjee/lib/jsr311-api-1.0.jar
   - /opt/FJSVisjee/lib/jettison-1.0.1.jar
   - /opt/FJSVisjee/lib/jackson-asl-0.9.4.jar

   Run the following command to modify the names:

```
# cd /opt/FJSVisjee/lib/ <ENTER>
# mv jersey-bundle-1.0.3.1.jar  jersey-bundle-1.0.3.1.jar.backup <ENTER>
# mv jsr311-api-1.0.jar         jsr311-api-1.0.jar.backup <ENTER>
# mv jettison-1.0.1.jar         jettison-1.0.1.jar.backup <ENTER>
# mv jackson-asl-0.9.4.jar      jackson-asl-0.9.4.jar.backup <ENTER>
```

4. Restart the CEP service.

```
# cepstartserv <ENTER>
```

5. Set permissions for the Terracotta license file (terracotta-license.key) to enable the engine execution user to read it.

6. Edit the Terracotta Collaboration setup file. Refer to "9.3.2 Terracotta Collaboration Setup File" for details.

4.4.4 Setup of RDB Collaboration

Complex Event Processing of BDCEP can use RDB Collaboration to reference a relational database that is external to the CEP Server. To perform RDB collaboration, setup of RDB collaboration is also required.

BDCEP can collaborate with the relational databases supported by the JDBC driver of the products listed below. Note that in the given product names. *x* can be any number and *X* can be any letter.

- Symfoware Server V10.*x.xX* (V10.1.0A or later)
- Symfoware Server (Open Interface) V12.*x.x* (V12.0.0 or later)
- PostgreSQL JDBC Driver Version 9.*x-xxxx* (JDBC 4) (Version 9.3-1100 or later)

Setup procedure

1. On the CEP Server, install and set up a JDBC driver suited to the relational database you want to connect to. The CEP engine runs as a client of the relational database. Refer to the manual provided with the selected JDBC driver for details.

2. Edit the RDB Collaboration setup file. Refer to "9.4.1 RDB Collaboration Setup File" for details.

Note

If Symfoware Server with the native interface (not the PostgreSQL extended interface) is used for collaboration, set 16 or higher as the number of simultaneous processes in the relational database environment.

4.4.5 Setup of the CEP Engine

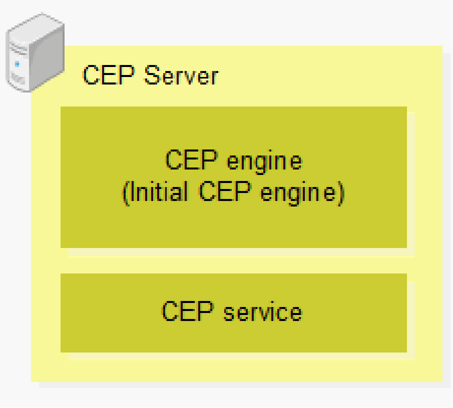
This section explains the setup of the CEP engine.

- Status Immediately after Installation
- Changing CEP Engine Settings
- Creating a New CEP Engine

4.4.5.1 Status Immediately after Installation

Immediately after installation, the CEP service that manages the CEP engine should be running, and one immediately usable CEP engine will be created. This CEP engine is hereafter referred to as the "initial CEP engine" when it is necessary to distinguish it from other CEP engines. The initial CEP engine will have the stopped status.

Figure 4.1 Status immediately after installation



This initial CEP engine has the settings shown below. This initial CEP engine can be used to deploy and check the operation of a sample application.

Item	Value to be set
CEP engine name	The name specified at installation. This is "CepEngine" if the specification was omitted at installation.
Logging type	Not set (Logging not used).
Directory name	Same as above
Number of open logging files	Same as above.
Logging cycle time	Same as above.
Socket adapter port	9600
JVM High-speed Filter options (*1)	Maximum value of memory allocation pool: 2048 MB Initial value of memory allocation pool: 512 MB Maximum value of permanent generation area: 192 MB
JVM Complex Event Processing options (*1)	Same as above.

*1: Settings relating to memory used by the CEP engine. Refer to "6.3.4.1 Tuning JVM Options" for information on each parameter.

Below is the engine configuration file used in initial CEP engine creation.

```
/etc/opt/FJSVcep/cep/sample_eng.xml
```

Refer to "9.1.1 Engine Configuration File" for information on how to view the contents of the engine configuration file.

4.4.5.2 Changing CEP Engine Settings

When using logging in the initial CEP engine, the CEP engine settings must be changed.

The two methods below are used to change the CEP engine settings, and the items that can be changed will vary depending on the method used.

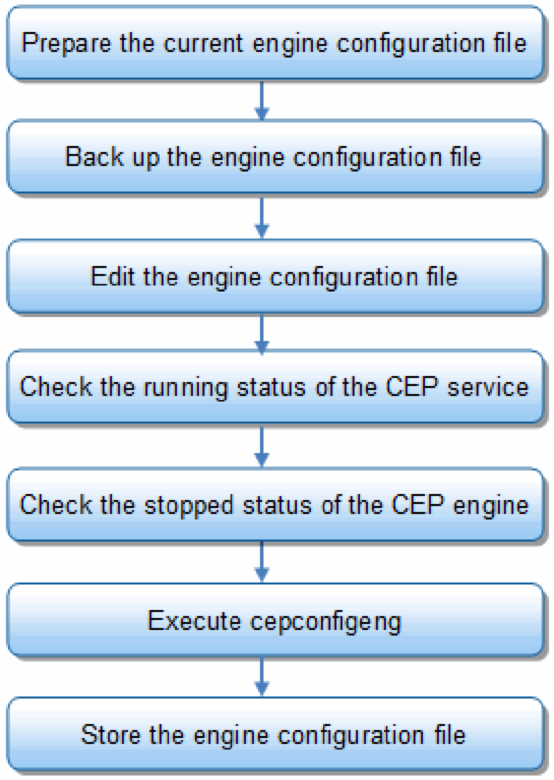
Method for changing settings	Items that can be changed
Using cepconfigeng	- Logging settings - Port number used in socket communication
Using cepsetjvmopt	- Memory used by CEP engine

Changing settings using cepconfigeng is explained below.

Refer to "6.3.4.1 Tuning JVM Options" for information on changing settings using cepsetjvmopt.

Changing settings using cepconfigeng

Below is a flowchart for changing settings using cepconfigeng.


Prepare the current engine configuration file
Back up the engine configuration file
Edit the engine configuration file
Check the running status of the CEP service
Check the stopped status of the CEP engine
Execute cepconfigeng
Store the engine configuration file

The procedure for changing CEP engine settings is explained below. For these tasks, log in as a superuser to execute the commands.

1. **Prepare the current engine configuration file.**

   A CEP engine that is not to be changed must also be defined in the engine configuration file to be specified at cepconfigeng command execution. Therefore, define the changes based on the current stored engine configuration file.

   The engine configuration file used in the creation of the initial CEP engine is stored in the following location:

   ```
   /etc/opt/FJSVcep/cep/sample_eng.xml
   ```

   Example

   Example of preparing the engine configuration file used in the creation of the initial CEP engine

   ```
   # cp /etc/opt/FJSVcep/cep/sample_eng.xml /etc/opt/FJSVcep/Engine.xml<ENTER>
   ```

Note

If the engine configuration file to be specified at cepconfigeng command execution does not include a created CEP engine definition, cepconfigeng will delete the CEP engine which is not specified in the engine configuration file.

2. **Back up the engine configuration file.**

   Before editing the current engine configuration file, create a backup of the engine configuration file. Always create a backup to avoid losing definition information erroneously.

Example

Example of backing up the engine configuration file

```
# cp /etc/opt/FJSVcep/Engine.xml /etc/opt/FJSVcep/Engine.bak.xml<ENTER>
```

3. **Edit the engine configuration file.**

   Edit the engine configuration file using a command such as vi, and then define the settings for the target CEP engine.

   Refer to "9.1.1 Engine Configuration File" for information on the format of the engine configuration file.

Example

Definition example of enabling logging for the initial CEP engine

The definition example below shows "enabling logging of the CEP Server (output destination is the engine log)" for the initial CEP engine immediately after installation.

The "CepEngine" that is displayed in the definition depends upon the CEP engine name specified at installation.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<subSystemConfig xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1">
  <engineConfig id="CepEngine">
    <logging>
      <type>file</type>
    </logging>
    <socketAdapterPort>9600</socketAdapterPort>
  </engineConfig>
</subSystemConfig>
```

4. **Check the running status of the CEP service.**

   The CEP service must be running to execute cepconfigeng. Use cepdispserv to check the status of the CEP service. Refer to "8.5 cepdispserv" for details.

   If the CEP service is not running, use cepstartserv to start it. Refer to "8.10 cepstartserv" for details.

Example

Example of using cepdispserv to check the running status of the CEP service

Execute cepdispserv to check that the content below is output.

The operating process number will be displayed where "*nnnn*" is displayed.

Generated CEP engine names are displayed in place of *cepEngine*.

```
# cepdispserv<ENTER>
(...)
Interstage Java EE DAS           started
(...)
```

```
Interstage Java EE Node Agent    started
(...)
CEPAgentIJServerCluster running
cepEngine_flt not running
cepEngine_cep not running
(...)
Status            : Running
(...)
jsvc (pid nnnn nnnn) is running...
(...)
pg_ctl: server is running (PID: nnnn)
(...)
Command cepdispserv executed successfully.
```

Information

The CEP service entity

A CEP service is a service made up of multiple processes.

5. **Check the stopped status of the CEP engine.**

   The target CEP engine must be stopped for cepconfigeng to change the settings of the CEP engine. Use cepdispeng to check the status of the engine. Refer to "8.4 cepdispeng" for details.

   If the CEP engine is running, use cepstopeng to stop the CEP engine. Refer to "8.11 cepstopeng" for details.

Example

Example of using cepdispeng to check the stopped status of the initial CEP engine

Execute cepdispeng to check the contents of the initial CEP engine output, as shown below.

The "CepEngine" that is displayed in the command execution example and output example depends upon the CEP engine name specified at installation.

```
# cepdispeng -e CepEngine<ENTER>
engineId         :CepEngine
port             :9600
status_filter    :STOP
status_cep       :STOP
Command cepdispeng executed successfully.
```

6. **Execute cepconfigeng.**

   Specify the edited engine configuration file and execute cepconfigeng. When the command is executed, confirmation of the change is requested, type "y" to continue execution. Execution of the command can be canceled by typing "n" or "q". Refer to "8.2 cepconfigeng" for details.

Example

Example of executing cepconfigeng

Below is an example of specifying "/etc/opt/FJSVcep/Engine.xml" as the edited engine configuration file.

```
# cepconfigeng -f /etc/opt/FJSVcep/Engine.xml<ENTER>
Are you sure you want to change the CEP Engine configuration? [y,n,q]:y<ENTER>
Command cepconfigeng executed successfully.
```

7. **Store the engine configuration file.**

   The engine configuration file specified in cepconfigeng will be required in future for further additions or deletions of CEP engines or for changing settings, so store it in a safe place. Consider creating a backup on external media, as required.

   Note

   The backup creation method depends upon the management policy of the system. Always check with the system administrator.

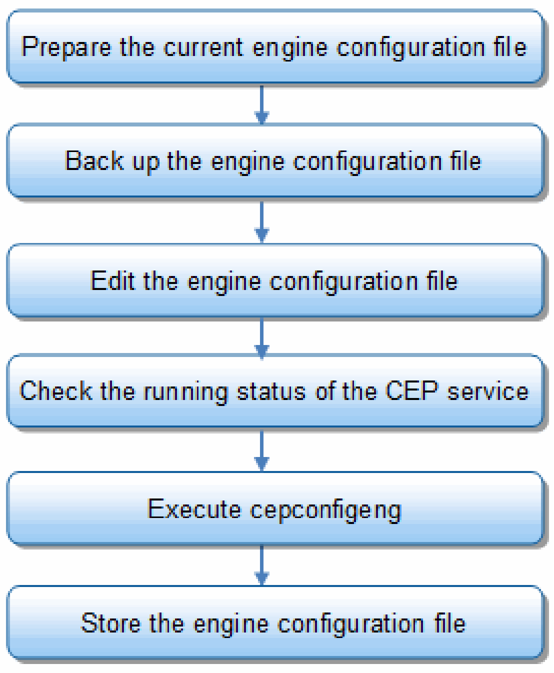
4.4.5.3 Creating a New CEP Engine

In general, only one CEP engine operates, but there may be times when multiple CEP engines need to be provided, such as when a development environment is being divided for multiple development groups.

BDCEP allows a maximum of five CEP engines to be created for one CEP Server. This section explains how to create a new CEP engine using cepconfigeng.

Creating a new CEP engine using cepconfigeng

Below is a flowchart for creating a new CEP engine using cepconfigeng.


Prepare the current engine configuration file
Back up the engine configuration file
Edit the engine configuration file
Check the running status of the CEP service
Execute cepconfigeng
Store the engine configuration file

The procedure for creating a new CEP engine is explained below. For these tasks, log in as a superuser to execute the commands.

1. **Prepare the current engine configuration file.**

   An existing CEP engine must also be defined in the engine configuration file to be specified at cepconfigeng command execution. Therefore, define the settings of the CEP engine to be added based on the current stored engine configuration file.

   Point

   If an operation has been performed previously according to the execution example in "4.4.5.2 Changing CEP Engine Settings", the current engine configuration file will be the following file:

```
/etc/opt/FJSVcep/Engine.xml
```

Note

If the engine configuration file to be specified at cepconfigeng command execution does not include an existing CEP engine definition, cepconfigeng will delete the CEP engine which is not specified in the engine configuration file.

2. **Back up the engine configuration file.**

   Before editing the current engine configuration file, create a backup of the engine configuration file. Always create a backup to avoid losing definition information erroneously.

Example

Example of backing up the engine configuration file

```
# cp /etc/opt/FJSVcep/Engine.xml /etc/opt/FJSVcep/Engine.bak.xml<ENTER>
```

3. **Edit the engine configuration file.**

   Edit the engine configuration file using a command such as vi, and then define the settings for the new CEP engine to be added.

   Use a CEP engine name that does not duplicate the name of another CEP engine. Refer to "9.1.1 Engine Configuration File" for information on the format of the engine configuration file.

Example

Example of creating a new CEP engine called "NewCepEngine"

Create a new CEP engine with the settings below, in addition to the initial CEP engine changed according to the example of executing changes to the settings of a CEP engine in "4.4.5.2 Changing CEP Engine Settings".

Item	Value to be set	Description
CEP engine name	NewCepEngine	Use a CEP engine name that is unique.
Logging type	Logging is not used, so do not set a value.	
Directory name	Same as above	
Number of open logging files	Same as above.	
Logging cycle time	Same as above.	
Socket adapter port	Socket communication is not performed, so do not set a value.	If not set, only a SOAP adapter or HTTP adapter can be used.

Below is an example of defining an engine configuration file.

The part from "`<engineConfig id="NewCepEngine">`" to "`</engineConfig>`" in lines 9 and 10 is the definition of the new CEP engine to be added.

The existing CEP engine definition remains as is. The "CepEngine" that is displayed in the definition depends upon the CEP engine name specified at installation.

```
1   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2   <subSystemConfig xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1">
3     <engineConfig id="CepEngine">
4       <logging>
5         <type>file</type>
6       </logging>
7       <socketAdapterPort>9600</socketAdapterPort>
```

```
  </engineConfig>
  <engineConfig id="NewCepEngine">
  </engineConfig>
</subSystemConfig>
```

4. **Check the running status of the CEP service.**

   The CEP service must be running to execute cepconfigeng. Use cepdispserv to check the status of the CEP service. Refer to "8.5 cepdispserv" for details.

5. **Execute cepconfigeng.**

   Specify the edited engine configuration file and execute cepconfigeng. When the command is executed, confirmation of the change is requested, so type "y" to continue execution. Execution of the command can be canceled by typing "n" or "q". Refer to "8.2 cepconfigeng" for details.

   Example

   Example of executing cepconfigeng

   Below is an example of specifying "`/etc/opt/FJSVcep/Engine.xml`" as the edited engine configuration file.

   ```
   # cepconfigeng -f /etc/opt/FJSVcep/Engine.xml<ENTER>
   Are you sure you want to change the CEP Engine configuration? [y,n,q]:y<ENTER>
   Command cepconfigeng executed successfully.
   ```

6. **Store the engine configuration file.**

   The engine configuration file specified in cepconfigeng will be required in future for further additions or deletions of CEP engines or for changing settings, so store it in a safe place. Consider creating a backup on external media, as required.

   Note

   The backup creation method depends upon the management policy of the system. Always check with the system administrator.

4.5 Canceling Setup

This section explains how to cancel the setup of the features set up in "4.4 Setup".

- Deleting a CEP Engine
- Canceling RDB Collaboration
- Canceling Terracotta Collaboration
- Canceling Hadoop Collaboration

4.5.1 Deleting a CEP Engine

This section explains how to delete a CEP engine.

Deleting a CEP engine that is no longer required allows the system resources such as memory and disk that were used by that CEP engine to be used for other purposes. This section explains how to delete a CEP engine using cepconfigeng.

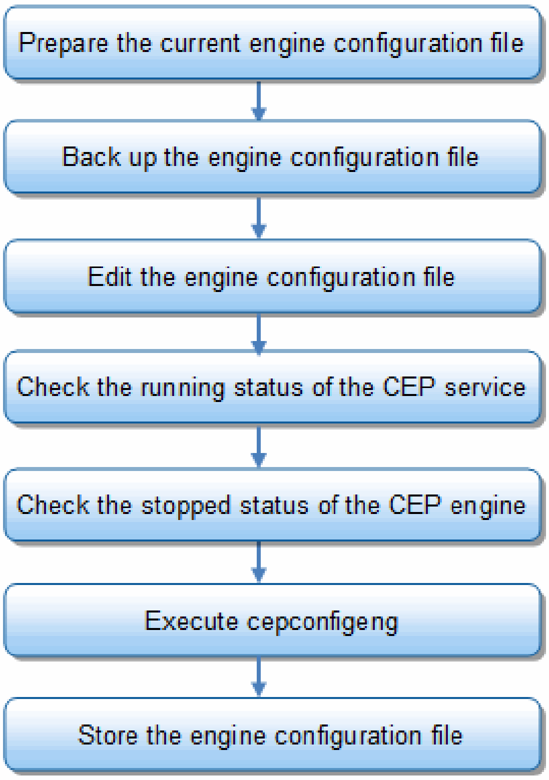
Note

When a CEP engine is deleted, definition information deployed to the CEP engine (such as rule definitions) will also be deleted. The engine log and resource log of the CEP engine will be deleted as well. Take a backup as required. Refer to "6.3.2 Backup and Restore" for details.

Deleting a CEP engine using cepconfigeng

Below is a flowchart for deleting a CEP engine using cepconfigeng.

Figure 4.2 Flowchart for deleting a CEP engine


Prepare the current engine configuration file
Back up the engine configuration file
Edit the engine configuration file
Check the running status of the CEP service
Check the stopped status of the CEP engine
Execute cepconfigeng
Store the engine configuration file

The procedure for deleting a CEP engine is explained below. For these tasks, log in as a superuser to execute the commands.

1. **Prepare the current engine configuration file.**

   An existing CEP engine must also be defined in the engine configuration file to be specified at cepconfigeng command execution. Therefore, define the settings of the CEP engine to be deleted based on the current stored engine configuration file.

   Point

   If an operation has been performed previously according to the execution example in "4.4.5.3 Creating a New CEP Engine", the current engine configuration file will be the following file:

   ```
   /etc/opt/FJSVcep/Engine.xml
   ```

   Note

   If the engine configuration file to be specified in cepconfigeng does not include an existing CEP engine definition, cepconfigeng will delete the CEP engine which is not specified in the engine configuration file.

2. **Back up the engine configuration file.**

   Before editing the current engine configuration file, create a backup of the engine configuration file. Always create a backup to avoid losing definition information erroneously.

Example

Example of backing up the engine configuration file

```
# cp /etc/opt/FJSVcep/Engine.xml /etc/opt/FJSVcep/Engine.bak.xml<ENTER>
```

3. **Edit the engine configuration file.**

   Edit the engine configuration file using a command such as vi, and then comment out or delete the definition for the CEP engine to be deleted.

   Refer to "9.1.1 Engine Configuration File" for information on the format of the engine configuration file.

Example

Definition example of deleting a CEP engine called "NewCepEngine"

In this example, the CEP engine called "NewCepEngine" that was created according to the execution example in "4.4.5.3 Creating a New CEP Engine" will be deleted.

Below is a definition example of the engine configuration file.

Note that the definition of the relevant part in lines 9 to 12 is not simply deleted. Instead, the part from "`<engineConfig id="NewCepEngine">`" to "`</engineConfig>`" is commented out. Commenting out allows you to thoroughly check the deletion range while you work. The "CepEngine" that is displayed in the definition depends upon the CEP engine name specified at installation.

```
 1   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 2   <subSystemConfig xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1">
 3     <engineConfig id="CepEngine">
 4       <logging>
 5         <type>file</type>
 6       </logging>
 7       <socketAdapterPort>9600</socketAdapterPort>
 8     </engineConfig>
 9   <!--
10     <engineConfig id="NewCepEngine">
11     </engineConfig>
12   -->
13   </subSystemConfig>
```

4. **Check the running status of the CEP service.**

   The CEP service must be running to execute cepconfigeng. Use cepdispserv to check the status of the CEP service. Refer to "8.5 cepdispserv" for details.

5. **Check the stopped status of the CEP engine.**

   The target CEP engine must be stopped for cepconfigeng to be used to delete the CEP engine. Use cepdispeng to check the status of the engine. Refer to "8.4 cepdispeng" for details.

6. **Execute cepconfigeng.**

   Specify the edited engine configuration file and execute cepconfigeng. When the command is executed, confirmation of the change is requested, so type "y" to continue execution. Execution of the command can be canceled by typing "n" or "q". Refer to "8.2 cepconfigeng" for details.

Example

Example of executing cepconfigeng

Below is an example of specifying "`/etc/opt/FJSVcep/Engine.xml`" as the edited engine configuration file.

```
# cepconfigeng -f /etc/opt/FJSVcep/Engine.xml<ENTER>
Are you sure you want to change the CEP Engine configuration? [y,n,q]:y<ENTER>
Command cepconfigeng executed successfully.
```

7. **Store the engine configuration file.**

   The engine configuration file specified in cepconfigeng will be required in future for further additions or deletions of CEP engines or for changing settings, so store it in a safe place. Consider creating a backup on external media, as required.

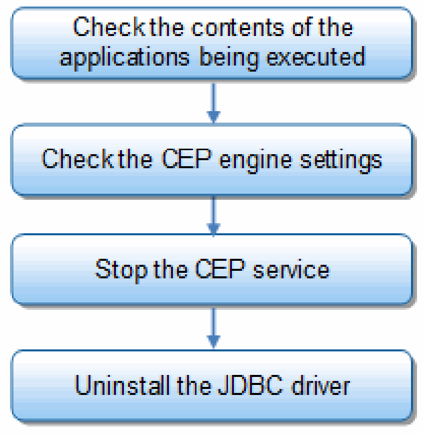
   Note

   The backup creation method depends upon the management policy of the system. Always check with the system administrator.

4.5.2 Canceling RDB Collaboration

Below is a flowchart for canceling RDB collaboration.

Figure 4.3 Flowchart for canceling RDB collaboration


Check the contents of the applications being executed
Check the CEP engine settings
Stop the CEP service
Uninstall the JDBC driver

1. **Check the contents of the applications being executed.**

   Make inquiries to the application maintenance staff to ensure that the applications being executed do not require RDB collaboration.

2. **Check the CEP engine settings.**

   Check if an RDB reference definition has been deployed to an existing CEP engine. If an RDB reference definition has been deployed, undeploy the RDB reference definition.

3. **Stop the CEP service.**

   Stop the CEP service that is running. The stop method is shown below. Log in as a superuser to execute the command.

```
# cepstopserv<ENTER>
```

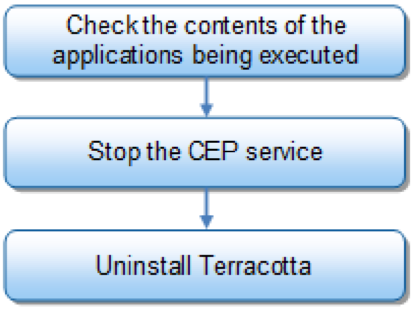
4. **Uninstall the JDBC driver.**

   Uninstall the JDBC driver. Refer to the manual provided with the JDBC driver for information on how to uninstall the driver.

4.5.3 Canceling Terracotta Collaboration

Below is a flowchart for canceling Terracotta collaboration.

Figure 4.4 Flowchart for canceling Terracotta collaboration



1. **Check the contents of the applications being executed.**

   Make inquiries to the application maintenance staff to ensure that the applications being executed do not require Terracotta collaboration.

   Information

   If an application is to use Terracotta collaboration to reference the Terracotta cache, specify vdw:ehcache() in the CREATE WINDOW statement in the complex event processing rule definition. Refer to "5.4.4.3.3 Using Terracotta cache" for details.

2. **Stop the CEP service.**

   Stop the CEP service that is running. The stop method is shown below. Log in as a superuser to execute the command.

```
# cepstopserv<ENTER>
```

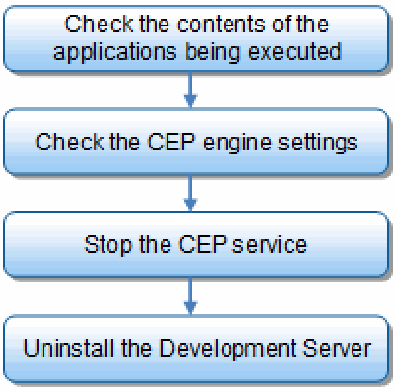
3. **Uninstall Terracotta.**

   Uninstall Terracotta. Refer to the Interstage Terracotta BigMemory Max manual for information on how to uninstall it.

4.5.4 Canceling Hadoop Collaboration

Below is a flowchart for canceling Hadoop collaboration.

Figure 4.5 Flowchart for canceling Hadoop collaboration


Check the contents of the applications being executed
Check the CEP engine settings
Stop the CEP service
Uninstall the Development Server

1. **Check the contents of the applications being executed.**

   Make inquiries to the application maintenance staff to ensure that the applications being executed do not perform Hadoop collaboration. If there is an application that is to perform Hadoop collaboration, consider modifying the application.

   Information

   An application uses the two methods below to use Hadoop collaboration. Refer to "Chapter 5 Development" for details.

- Specifying "`true`" in the useLogging item in the event type definition
- Using the "`@LoggingListener`" annotation in the complex event processing rule definition

2. **Check the CEP engine settings.**

   Check the contents of the engine configuration file used in the settings of the existing CEP engine to ensure the existing CEP engine is not specifying a Hadoop collaboration setting. If there is a CEP engine performing Hadoop collaboration, consider changing the settings of the CEP engine.

   Information

   If an operation has been performed previously according to "4.4.5 Setup of the CEP Engine", the current engine configuration file will be the following file:

   ```
   /etc/opt/FJSVcep/Engine.xml
   ```

   To perform Hadoop collaboration, "`bdpp`" will be specified in the "`type`" element under the "`logging`" element. Refer to "9.1.1 Engine Configuration File" for information on the format of the engine configuration file.

3. **Stop the CEP service.**

   Stop the CEP service that is running. The stop method is shown below. Log in as a superuser to execute the command.

   ```
   # cepstopserv<ENTER>
   ```

4. **Uninstall the Development Server.**

   Uninstall the Development Server. Refer to the Interstage Big Data Parallel Processing Server manual for information on how to uninstall it.

4.6 Uninstallation

This section explains how to uninstall BDCEP.

4.6.1 Pre-uninstallation Procedure

This section explains the tasks required before uninstalling BDCEP.

4.6.1.1 Stopping Event Sending

Stop event sending to the CEP Server.

4.6.1.2 Backing up User Assets

Back up user assets. Refer to "6.3.2 Backup and Restore" for information on how to back up user assets such as definition information.

4.6.1.3 Stopping the CEP Service

Stop the CEP service that is running. The stop method is shown below. Log in as a superuser to execute the command.

```
# cepstopserv<ENTER>
```

4.6.1.4 Deleting Updates

If the following updates of BDCEP and of the software bundled with BDCEP have been applied in UpdateSite format, delete the updates:

- Interstage Big Data Complex Event Processing Server
- Interstage Application Server Enterprise Edition (64 bit)

See

Refer to the Help in "UpdateAdvisor (middleware)" and the update information files of each update for details.

4.6.2 Uninstallation Procedure

This section explains the uninstallation procedure.

From the following two types, select the uninstallation method that best suits the method of use. If uninstallation using multi-user mode, check that the operations of other users will not affect the uninstallation:

Attended uninstallation

Use 'attended uninstallation' to execute the uninstallation with querying to check that uninstallation is to be executed and then executes uninstallation.

Unattended uninstallation

Use 'unattended uninstallation' to execute the uninstallation with no querying from the Uninstaller.

Note

With uninstallation, the packages installed by BDCEP will be uninstalled. Specific installed packages and features cannot be selected for uninstallation. Some packages will be uninstalled manually.

In addition, the resources under the installation destination directory will be deleted at uninstallation. Always save necessary resources before uninstallation. Refer to "6.3.2 Backup and Restore" for information on backup.

4.6.2.1 Attended Uninstallation

1. Log in as a superuser.

```
$ su - <ENTER>
```

2. Load the installation DVD-ROM and execute the uninstallation shell script (uninstall.sh shell) stored on the DVD-ROM from any directory.

```
# mount /dev/deviceFileName dvdRomMountDir <ENTER>
# dvdRomMountDir/uninstall.sh <ENTER>
```

3. The product name will be displayed as shown below.

```
+------------------------------------------------------------+
| Interstage Big Data Complex Event Processing Server V1.1.0 |
|                                                            |
|                      Copyright 2012-2013 FUJITSU LIMITED  |
+------------------------------------------------------------+
```

4. Continuing from the display above, the uninstallation content will be displayed as shown below.
In *uninstallPackages*, all of the packages to be uninstalled will be displayed delimited by spaces.

```
Uninstallation packages:
  uninstallPackages
```

5. Check the content and, to start uninstalling the packages, type "y" and press the Enter key.
To stop uninstallation, type "q" and press the Enter key.
After typing "y", BDCEP uninstallation will start. Uninstallation can be stopped by typing "q".

```
All files/directories under the installation destination directory will be deleted.
Please back up the required resources before uninstalling.

Do you want to proceed with the uninstallation? [y,q]:
```

6. When uninstallation has completed normally, the message below will be output. Perform the post-uninstallation procedure to continue.

```
The uninstallation processing completed successfully.
```

4.6.2.2 Unattended Uninstallation

1. Log in as a superuser.

```
$ su - <ENTER>
```

2. Load the installation DVD-ROM and, from any directory, execute the uninstallation shell script (uninstall.sh shell) stored on the DVD-ROM, with the -s option added.

```
# mount /dev/deviceFileName dvdRomMountDir <ENTER>
# dvdRomMountDir/uninstall.sh -s<ENTER>
```

3. When uninstallation has completed normally, the message below will be output.

```
The uninstallation processing completed successfully.
```

4.6.3 Post-uninstallation Procedure

This section explains the tasks after uninstallation.

4.6.3.1 Uninstalling FJSVod

The uninstaller does not uninstall the FJSVod package when the following product is installed on CEP Server.

- Systemwalker Centric Manager (Management Server)

Below are manual uninstallation steps for the FJSVod package.

1. Check product usage

   Check if the following product is installed. If it is installed, do not uninstall the FJSVod package.

   - Systemwalker Centric Manager (Management Server)

2. Uninstall the FJSVod package

   Execute rpm as a superuser to uninstall the FJSVod package.

```
# /bin/rpm -e --nodeps FJSVod <ENTER>
```

4.6.3.2 Uninstalling FJSVsmee64 and FJSVsclr64

The uninstaller does not uninstall the FJSVsmee64 and FJSVsclr64 packages, because these packages may be included in Fujitsu non-Interstage products such as Systemwalker Centric Manager.

If you want to uninstall these packages manually, execute following commands as a superuser.

```
# /bin/rpm -e --nodeps FJSVsmee64 <ENTER>
# /bin/rpm -e --nodeps FJSVsclr64 <ENTER>
```

4.6.3.3 Removing Environment Variables

Delete the path below from the PATH environment variable of the CEP Server users.

```
/opt/FJSVcep/bin
```

Point

If the following files were created in the /etc/profile.d directory in "4.3.3.1 Setting Environment Variables", delete the created files:

- /etc/profile.d/FJSVcep.sh
- /etc/profile.d/FJSVcep.csh

4.6.3.4 Engine Execution User Specified at Installation

The engine execution user and the group to which the engine execution user belongs that are specified at installation will not be deleted at uninstallation. If the user is not required, delete the user.

Example

Below is an example of manually deleting the execution user called "isbdcep".

```
$ su - <ENTER>
# userdel isbdcep<ENTER>
```

Note

The user and group deletion method depends upon the management policy of the system. Always check with the system administrator.

4.6.3.5 Uninstall (middleware)

Installing BDCEP also installs "Uninstall (middleware)".

"Uninstall (middleware)" is a common tool for Fujitsu middleware products. It manages information about installed Fujitsu middleware products and starts the product uninstaller.

Perform the following procedure to uninstall this tool:

1. Start "Uninstall (middleware)", and check if any other Fujitsu middleware products exist on the system. Start the tool as follows:

   ```
   # /opt/FJSVcir/cir/bin/cimanager.sh -c <ENTER>
   ```

2. If there are no installed Fujitsu middleware products, run the following uninstallation command:

   ```
   # /opt/FJSVcir/bin/cirremove.sh <ENTER>
   ```

3. When "This software is a common tool of Fujitsu products. Are you sure you want to remove it? [y/n]:" is displayed, type "y" to continue with the uninstallation process.

   The uninstallation process completes in a few seconds.

4. After the uninstallation completes, the following directory and the files under it will still exist, so delete it.

   ```
   /var/opt/FJSVcir
   ```

4.6.4 If an Error Occurs during Uninstallation

If execution of the uninstaller fails, after responding in accordance with the message, re-execute the uninstall.sh shell.

Refer to Section 2.2, "Errors during Uninstallation" in *Troubleshooting* for details.

Information

Log file of the uninstaller

Below is the log file output by the uninstaller. It records detailed operation statuses in addition to the messages displayed in the windows and is referred to when trouble occurs.

```
/var/tmp/bdcep_uninstall.log
```

When uninstallation is successful, the log file will be deleted.

Chapter 5 Development

This chapter explains event processing using Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP"), from design to development and testing.

This chapter also explains the sample application.

5.1 Overview of BDCEP Event Processing

BDCEP event processing is broadly divided into the following two types:

- Event processing that uses logging to record input events in an external Hadoop system.
- Event processing that applies a series of rules to input events and either logs the processing results, sends them to an external Web service, or processes them using a user-developed Java class. This type of event processing also performs processes such as referencing external data (distributed cache, relational database) when applying the rules.

Deploying the following definition information allows the system to perform these series of operations. The sections that follow explain the details of this definition information:

- Event type definition (mandatory)
- Rule definition (optional)
- Master definition (optional)
- RDB reference definition (optional)
- SOAP listener definition (optional)

In addition to this definition information, using BDCEP to utilize the processing results of events will also require development assets, such as applications for analyzing logged events or processing results and Web services that operate after receiving the processing results of events.

5.2 List of Development Assets

The table below lists the development assets of BDCEP. Development assets are broadly divided into definition information, data, and collaboration applications. Definition information is deployed in the CEP engine for execution. Each of the others is provided (deployed) in its corresponding server.

Table 5.1 List of development assets

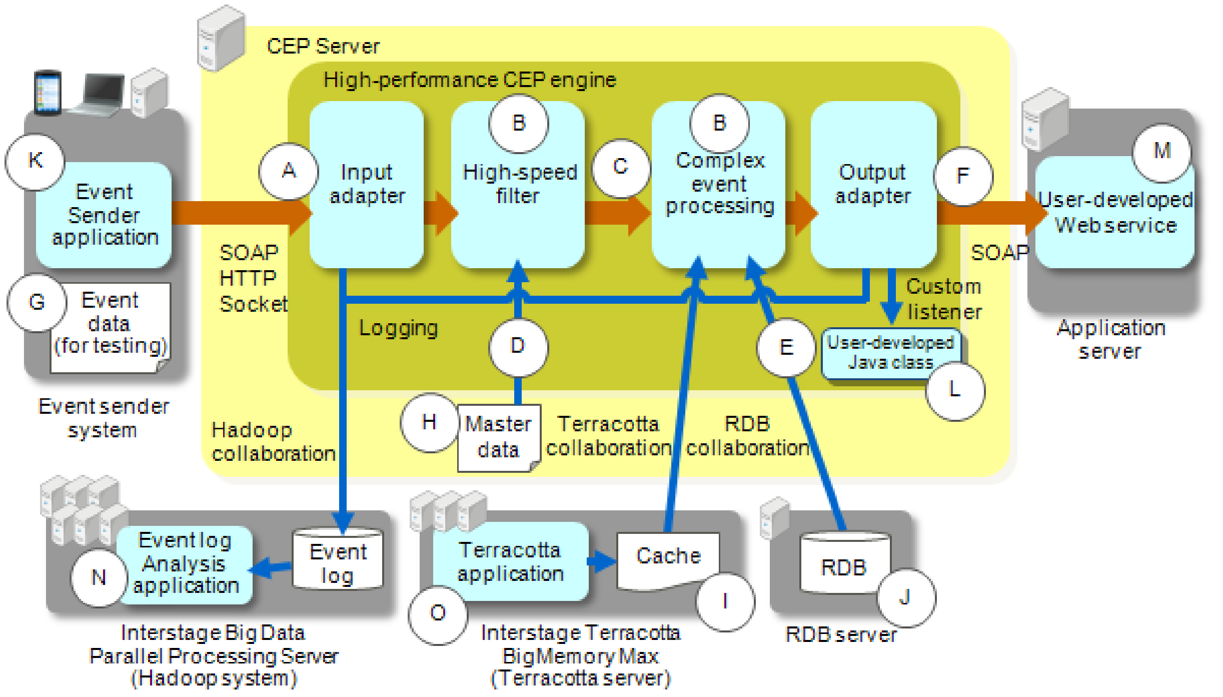
Development asset type	Development asset	Explanation	Deployment destination
Definition information	Event type definition	Define the format of the events the input adapter is to receive. Also define the log storage area specification and whether or not complex event processing is to be used. The events the input adapter is to receive can be in XML or CSV format.	CEP engine
	Rule definition	Define high-speed filter processing rules and complex event processing rules. Describe these using filter rule language (IF-THEN format) and complex event processing rule language (SQL format), respectively. The processing results of complex event processing rules can be sent to an external application by SOAP communication, accumulated on disks by logging, and processed according to a user-developed Java class by the custom listener.	

Development asset type	Development asset	Explanation	Deployment destination
	Event type definition (filtered events)	Define this if the items that make up the events will vary according to factors such as join and extraction processing of high-speed filter processing. Filtered events will be in CSV format.	
	Master definition	Create this if master referencing is to be performed in high-speed filter processing. This is the definition of the master data to be referenced.	
	RDB reference definition	Create this if RDB collaboration is to be performed using complex event processing. Define RDB connection information.	
	SOAP listener definition	Define the interface of the user-developed Web service to be used as the send destination when the processing results of complex event processing rules are to be sent to an external application using SOAP.	
Data	Event data (for testing)	This is event data to be sent to the CEP Server to check the operation of definition information.	Event sender system
	Master data (for the high-speed filter)	This is required separately if master referencing is to be performed in high-speed filter processing. Provide this in CSV file format on the CEP Server.	CEP Server
	Terracotta (*1) cache	This is required separately if Terracotta collaboration is to be performed. Provide this on the collaborating Terracotta server. A Terracotta application for update is required separately in order to store or update the data in a Terracotta cache.	Terracotta server
	Relational database (RDB)	This must be considered if RDB collaboration is to be performed. Provide this on the collaborating RDB server. RDB commands or an RDB application is required for storing and updating data in a relational database.	RDB server
Collaboration application	Event sender application	This is an application that sends events to the CEP engine. To use SOAP for sending events, provide a SOAP client application. This is not required if events are to be sent directly to the CEP engine using an existing system as the event sender. If a new event sender application needs developing, it must be done according to the device to be used as the sender. The event sender sample program bundled with the product can be used in the operation testing of rule definitions.	Event sender system
	User-developed Java class	This Java class receives the output of complex event processing rules via the custom listener and processes it. It is deployed to the CEP Server and called from a CEP engine.	CEP engine
	User-developed Web service	This is a Web service (SOAP application) that receives and controls event data sent by the output adapter. Deploy this in the application server that collaborates with CEP Server.	Application server

Development asset type	Development asset	Explanation	Deployment destination
	Event log analysis application	This is an application for analyzing event logs logged in a Hadoop system. Develop this using the Hadoop Java API. Deploy this in the Hadoop system and then execute it.	Hadoop system
	Terracotta application	This is an application, separate from the complex event processing rules, for updating the cache contents such as by initially storing data in a Terracotta cache. This is not required if an existing Terracotta cache is to be used.	Terracotta server

The following figure below shows the deployment destination of each development asset.

Figure 5.1 Deployment destinations of development assets



A: Event type definition
B: Rule definition (left: Filter rule, right: Complex event processing rule)
C: Event type definition (filtered events)
D: Master definition
E: RDB definition
F: SOAP listener definition

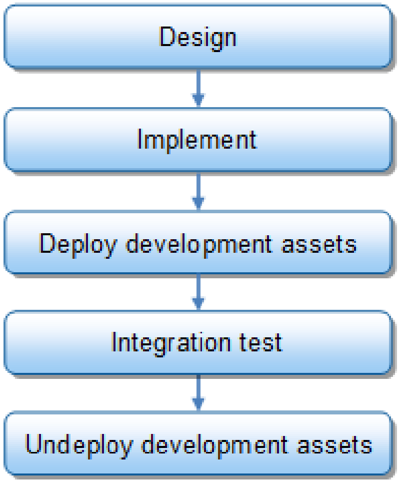
G: Event data (for testing)
H: Master data (for high-speed filter)
I: Terracotta cache
J: RDB

K: Event sender application
L: User-developed Java class (custom listener)
M: User-developed Web service
N: Event log analysis application
O: Terracotta application

5.3 Task Overview

The development flow is shown below.

Figure 5.2 Development flow



Development environment

Design and develop definition information using a developer's own local personal computer, and check the operation by deploying the definition information in a CEP engine on the CEP Server.

For the event data for testing or master data (CSV), use data extracted from an existing database or data generated using a tool such as Excel.

When developing a collaboration application, use a development environment that is suitable for the collaborating system or product.

5.4 Design (Definition File)

This section explains how to design each type of definition information.

5.4.1 Overview of Definition Information

This section provides an overview of items included in the following definition information:

- Event Type Definition
- Rule Definition
- Master Definition
- RDB Reference Definition
- SOAP Listener Definition

5.4.1.1 Event Type Definition

The table below provides an overview of event type definitions.

Table 5.2 Overview of event type definitions

Item	Description
Development asset ID	Specify an ID that is unique in the deployment destination CEP engine. A development asset ID for the event type definition is used as an event stream name in complex event processing rules.
Format	Specify the format of the event that is input to filter processing and complex event processing. Specify XML or CSV.

Item	Description
	Filtered events will only be in CSV format.
CSV column information	Set this item if the event format is CSV. Define column names and type information.
XML schema (*1)	Set this item if the event format is XML. Specify a schema (XML schema), which represents the event structure.
Root element (*1)	Specify an event root element name if the event type is XML. This will be one of the XML schema defined elements.
Use of logging	Specify whether to use Logging for received events that have not been processed by the high-speed filter in the input adapter.
Log storage area (*1)	Specify a storage area if Logging is used in the input adapter.
Use of complex event processing	Specify whether to use complex event processing.

*1: There is no need to consider this if the definition information is "event type definition (filtered)". Refer to "5.4.6 Designing an Event Type Definition (Filtered)" for information on this definition.

5.4.1.2 Rule Definition

The table below provides an overview of rule definitions.

Table 5.3 Overview of rule definitions

Item	Description
Development asset ID	Specify an ID that is unique in the deployment destination CEP engine.
Filter rule	Define high-speed filter rules. Multiple filter statements can be specified in a rule by specifying them consecutively. Use relevant development asset ID values in the following rule items: - ON statement event type Specify a development asset ID for the event type definition that corresponds to input events. - Master data Specify a development asset ID for the master definition. - Output expression event type alias Specify a development asset ID for the event type definition that corresponds to filtered events.
Complex event processing rule	Define rules for complex event type processing (complex event processing rules). Multiple complex event processing statements can be defined in a rule by delimiting them using semicolons. Use relevant development asset ID values in the following rule items: - Input event stream names Specify the development asset ID for the event type definition. - Database name Specify the development asset ID for the RDB reference definition. - SoapListener annotation Specify the development asset ID for the SOAP listener definition.

5.4.1.3 Master Definition

The table below provides an overview of master definitions.

Table 5.4 Overview of master definitions

Item	Description
Development asset ID	Specify an ID that is unique in the deployment destination CEP engine.
Schema file	Specify a file in which data file item names are defined.
Data file	Specify a directory path where data files are stored, or the data file path. If a directory is specified, all files stored in the specified directory excluding subdirectories are treated as the master data. Multiple data files can be specified and treated as single master data.
Use of skip	Specify whether to skip the first line of a data file.

5.4.1.4 RDB Reference Definition

The table below provides an overview of RDB reference definitions.

Table 5.5 Overview of RDB reference definitions

Item	Description
Development asset ID	Specify an ID that uniquely identifies the definition information.
Database system name	Specify the name of the database system. (*1)
Schema name	Specify the schema name of the database to be connected. (*1)
JDBC driver class	Specify the class of the JDBC driver. (*2)
Database URL	Specify the URL of the database to be connected.
Access ID	Specify the name of the user who will connect to the relational database.
Access password	Specify the password of the user who will connect to the relational database.
Cache retention period	Specify the time period for holding the RDB reference results in the cache.
Cache purge interval	Specify the time interval for flushing the cache of RDB reference results.

*1: Required when referencing Symfoware with the native interface.

*2: Required when referencing Symfoware (Open Interface) or PostgreSQL.

5.4.1.5 SOAP Listener Definition

The table below provides an overview of SOAP listener definitions.

Table 5.6 Overview of SOAP listener definitions

Item	Description
Development asset ID	Specify an ID that is unique in the deployment destination CEP engine.
Connection URL	Specify a connection URL of the user-developed Web service.
Namespace	Specify a namespace of the message (SOAP body content) to be sent to the user-developed Web service.
Namespace prefix	Specify a namespace prefix used in the message (SOAP body content) to be sent to the user-developed Web service.
Root element	Specify a root element name of the message (SOAP body content) to be sent to the user-developed Web service.

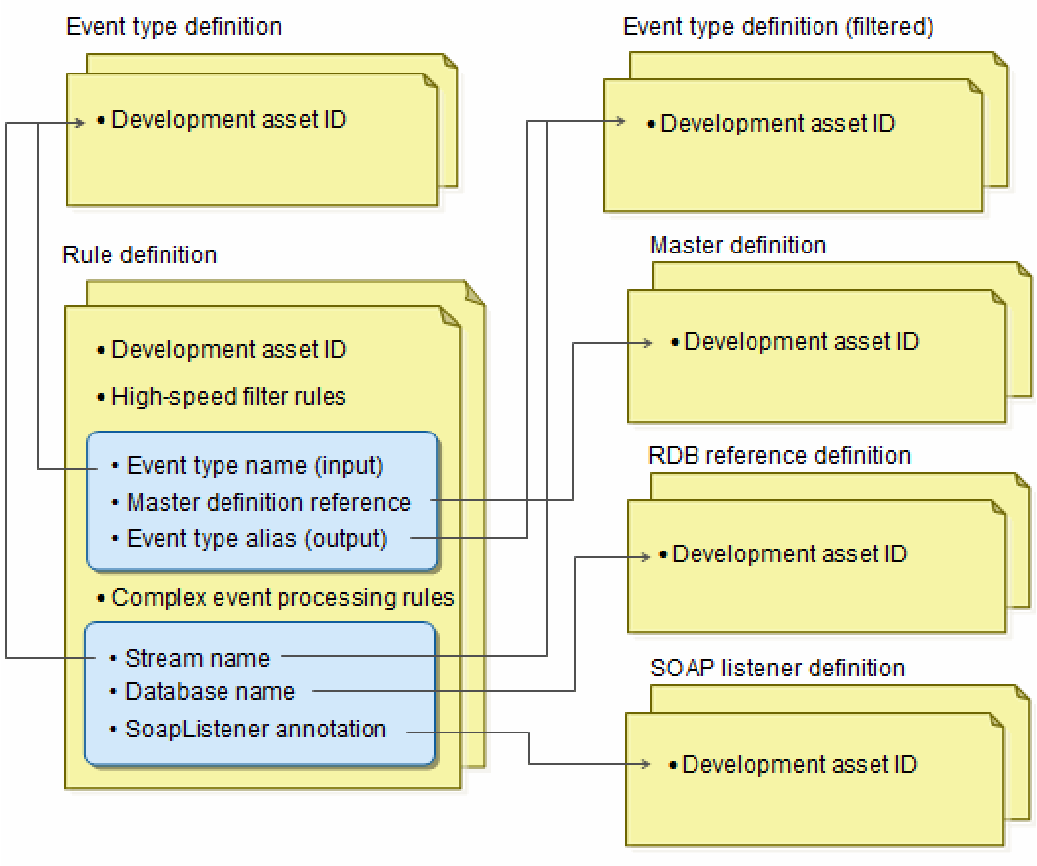
5.4.2 Association between the Development Asset ID and Definition Information

Each item of definition information has a development asset ID used to uniquely identify it as well as to associate it with other items of definition information.

When designing definition information, these development asset IDs must be managed.

Below is an example of the association between each development asset. The development asset IDs and rules described for each development asset must be defined so that they establish this association.

Figure 5.3 Association between development asset IDs and definition information



5.4.3 Designing an Event Type Definition

This section explains the points to consider when designing an event type definition.

5.4.3.1 Features of Input Events

Check details such as the data format and size of input events as well as their frequency of occurrence, as follows:

- If an existing system is to be used as the input event sender, check the event specifications of the existing system.
- Check the requirements relating to the amount of processing of the input events, such as the average input event size and the average number of events received per unit of time (for example, per hour or per second).
- In an event type definition, the two event data formats are XML and CSV format, as follows:
  - Select XML format or CSV format as appropriate.
  - Input events with other data formats such as binary format cannot be directly received, so consider converting the data format between the event sender and the CEP Server.

5.4.3.2 Recording and Analyzing Events

This section explains the points to consider when recording and accumulating events for analysis and other purposes.

To record events, use BDCEP Logging.

Accumulating events

To accumulate the events in the Hadoop system where they are being logged, consider the following points:

- Type and format of the events to be logged

  Consider what type of events to log and accumulate.

- Storage destination of the events to be logged

  The Hadoop system to be connected is set according to the engine configuration file of the CEP engine.

  In the Hadoop system to be connected, check for a path that can be used as a storage destination.

- Event accumulation capacity (free disk capacity required)

  The event storage destination will require enough free disk capacity to accumulate massive volumes of events.

  Check the average event size, number of recordings per unit of time, and accumulation period in order to consider the disk capacity required for recording and accumulating the events.

  If necessary, consider an expansion plan for the system to be used as the event storage destination.

Analyzing accumulated events

An application for analysis must be designed and developed in order to analyze the accumulated events.

Refer to "5.6.4 Designing an Event Log Analysis Application" for information on designing an application for analysis.

5.4.4 Designing a Rule Definition

A rule definition consists of two types of rules: the filter rules to be used in high-speed filter processing and the complex event processing rules to be used in complex event processing.

This section explains the following items, including considerations when designing rules and the creation procedures to use:

- High-speed Filter Processing
  - Considerations when creating filter rules
  - Processing pattern of filter rules
- Complex Event Processing
  - Considerations when creating complex event processing rules
  - Creation procedure for complex event processing rules
- Terracotta Collaboration
- RDB Collaboration
- SOAP Listener
- Custom Listener
- Logging Listener

5.4.4.1 High-speed Filter Processing

Define the filter rules to be used by the high-speed filter.

Considerations when creating filter rules

The items to consider when creating filter rules are as follows:

- Unit of rule creation

  Create a filter rule for each event type. Multiple filter rules cannot be defined simultaneously for a certain event type.

- Processing pattern of filter rules

  Select a suitable processing pattern from the ones described below, and then create a rule similar to the selected pattern.

  Refer to Chapter 2, "Filter Rule Language Reference" in the *Developer's Reference* for information on filter rules.

- Master data

  Master data is referenced from within filter rules. Some individual events may contain only limited information such as an ID or code in order to cut down on the volume of event communication. Writing rules to process events is difficult in that situation, so it is possible to use master data that can be referenced using the ID or code as a key. Using master data in this way allows rules to be created more easily.

  The master data must be created as files in CSV format by the user beforehand.

  Refer to "5.4.5 Designing a Master Definition" for information on designing master data.

- Memory usage for filter rules

  Using filter rules requires a large amount of memory. Refer to "3.3.1.1 Amount of Memory when Using High-speed Filter Rules" and "3.3.1.2 Amount of Memory when Master Data is used by the High-speed Filter" for information on the memory required.

Processing pattern of filter rules

By defining filter rules in the high-speed filter, the user can describe event extraction as well as extraction and join processing in combination with master data. The output of the high-speed filter is used directly as the input of complex event processing.

The processes performed by the high-speed filter are generally represented by the following four patterns and their combinations:

- Extraction process
- Extraction process using master data matching
- Join processing with master data
- Weighting processing of text

5.4.4.1.1 Extraction process

This processing pattern extracts from the input events those events that meet the conditions described in IF-THEN statements in the filter rules.

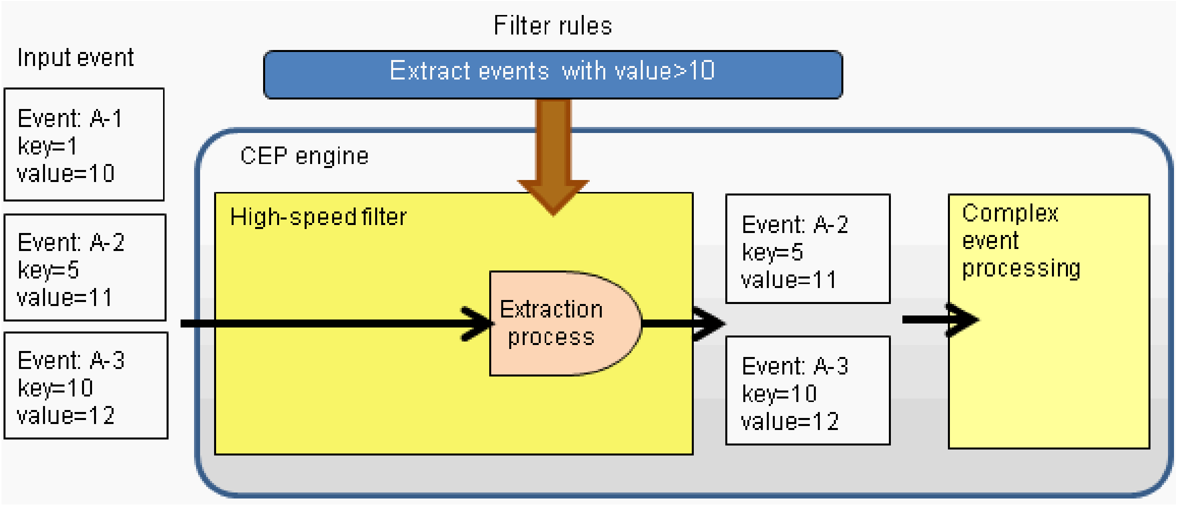
Consider using this processing pattern if only the events required are to be extracted from massive volumes of events.

Example

Example of an extraction process

This is an example of extracting events using the content of the events (`value`).

"`value > 10`" is defined as the extraction condition.



Rule to be created

The rule to be created in the example above is as follows:

```
on inputEventTypeID {
    if ($value > 10) then output() as inputEventTypeID;
}
```

- *`inputEventTypeID`* is the development asset ID of the target event type definition.

Information

Filtering the items in the events to be output

Items in the events to be output can also be filtered. Below is an example of outputting only "`key`" in the example shown above.

To output using a different format from that of the input events, a corresponding event type definition (filtered) will also be required.

Refer to Section 2.7, "Output Expression Format" in the *Developer's Reference* for details.

```
on inputEventTypeID {
    if ($value > 10) then output($key) as outputEventTypeID;
}
```

- *`inputEventTypeID`* is the development asset ID of the target event type definition.
- *`outputEventTypeID`* is the development asset ID of the event type definition (filtered) that represents the results of filtering the item.

5.4.4.1.2 Extraction process using master data matching

This processing pattern matches the relevant entries of master data (CSV files) on the basis of values such as those of the ID or code contained in the input events, and then extracts the events based on the values of the relevant entries.

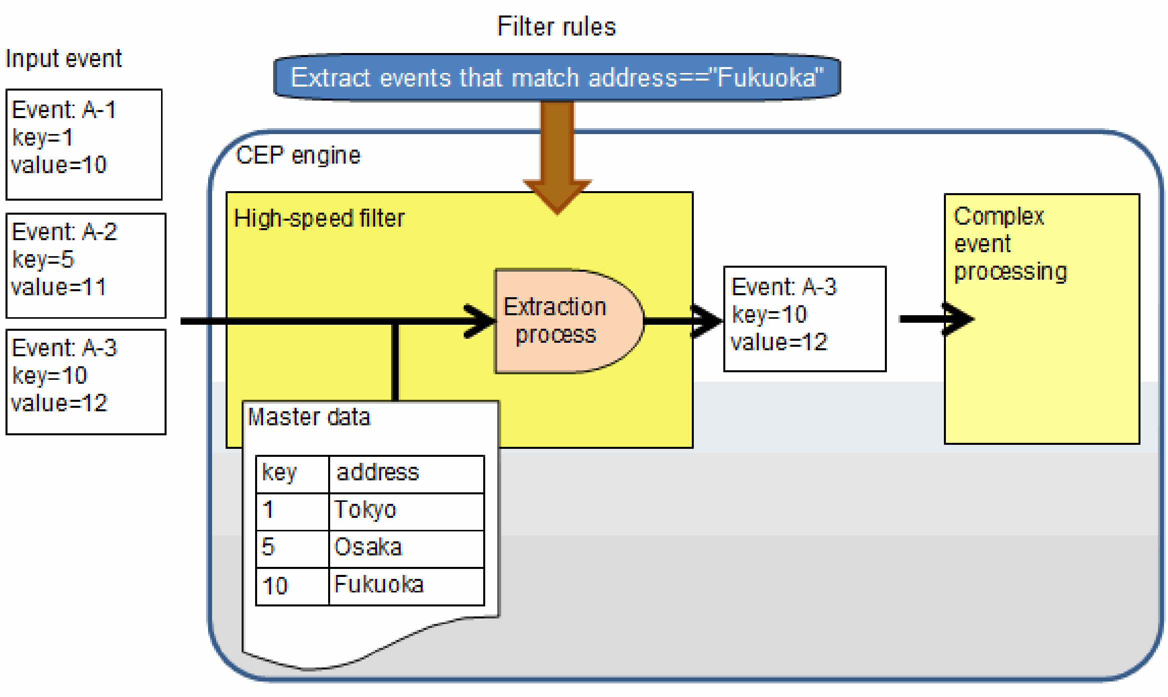
Consider using this processing pattern if only the events required are to be extracted from massive volumes of events but the events themselves do not contain the information required for extraction.

To perform the processing of this pattern, a master definition must also be designed.

Example

Example of the extraction process using master data matching

This is an example of referencing the relevant entries of master data based on the "key" contained in the events, and then extracting the events based on the values of "address" in the entries. "address==Fukuoka" is defined as the extraction condition.



key	address
1	Tokyo
5	Osaka
10	Fukuoka

Rule to be created

The rule to be created in the example above is as follows:

```
on inputEventTypeID {
    if (lookup("masterDefinitionID", $key == $key, string($address)) == "Fukuoka") then output() as
inputEventTypeID;
}
```

- *`inputEventTypeID`* is the development asset ID of the target event type definition.
- *`masterDefinitionID`* is the development asset ID of the master definition that corresponds to the master data to be referenced.
- To compare "`$address`" as a string, use "`string($address)`" to fetch the values.
- The left side of "`$key == $key`" is "`key`" of the input events and the right side is "`key`" of the master data.

Note

The master data information will not be assigned to the input events if they are only matched using "`lookup`". If join processing with master data is required, a join expression must be described. Refer to "5.4.4.1.3 Join processing with master data" below for information on join processing.

5.4.4.1.3 Join processing with master data

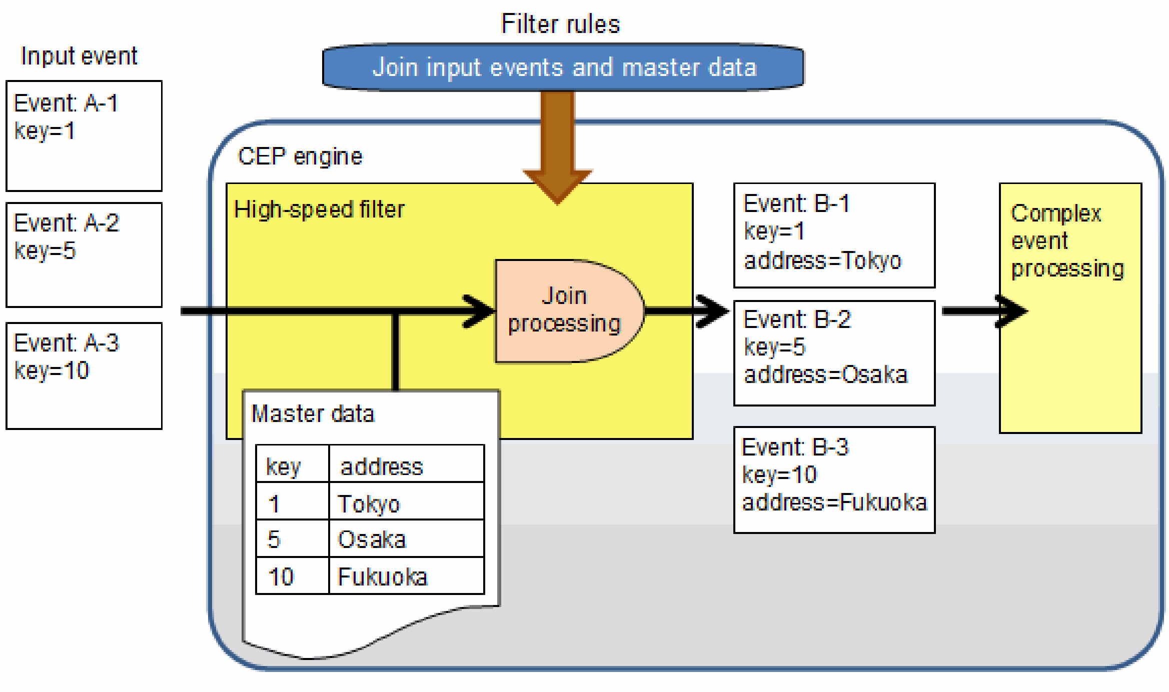
This processing pattern joins input events with master data. Consider using this processing pattern to assign the required data for the next complex event processing. It enables faster join processing than RDB referencing using complex event processing rules.

To pass the results of joining to complex event processing, an event type definition (filtered) corresponding to the join results will also be required.

Example

Example of join processing with master data

This is an example of joining the corresponding master data on the basis of "`key`" contained in the events, and then assigning "`address`" to the events.



Rule to be created

The rule to be created in the example above is as follows:

```
on inputEventTypeID {
    join("masterDefinitionID", $key == $key) output($key, "masterDefinitionID".$address) as
outputEventTypeID;
}
```

- *`inputEventTypeID`* is the development asset ID of the event type definition that is the rule target.
- *`masterDefinitionID`* is the development asset ID of the master definition that corresponds to the master data to be referenced.
- *`outputEventTypeID`* is the development asset ID of the event type definition (filtered) that represents the results of joining.

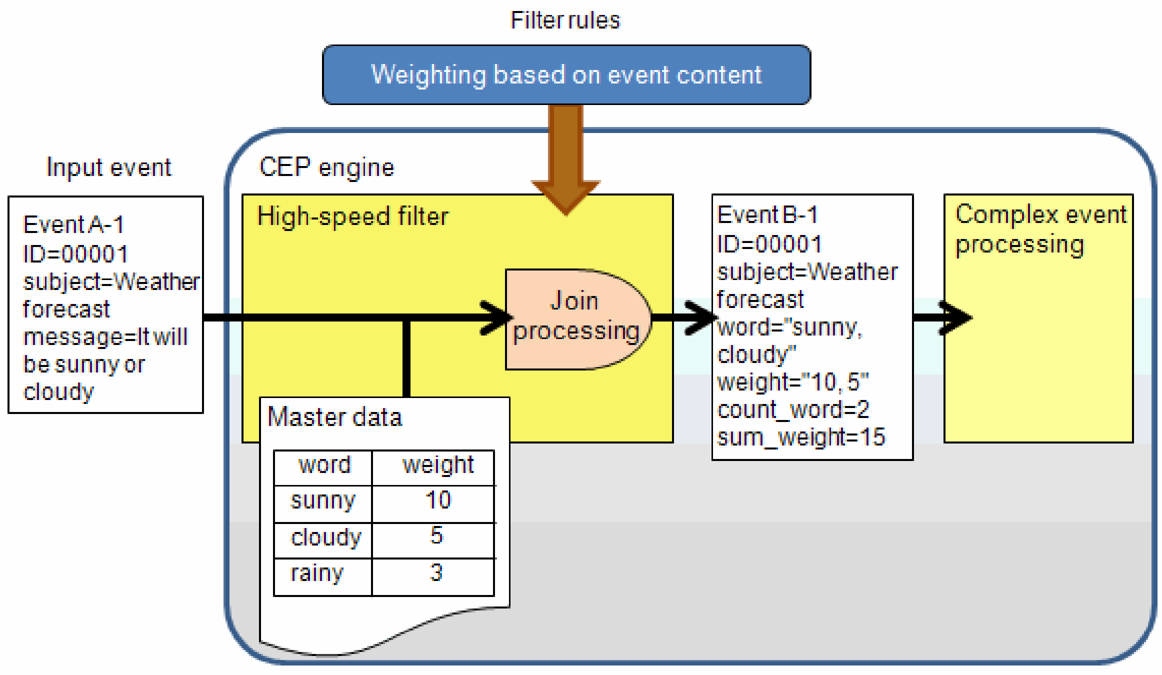
5.4.4.1.4 Weighting processing of text

This processing pattern can weight the text in input events by registering the weight of the keywords in the master data. This in turn allows applications including those that extract only those events with a total weighting that is above a threshold, and those that detect consecutively issued events that are above a threshold.

Example

Example of weighting processing of text

If the text contained in events contains search words that have been defined in the master data, assign the number of search words it contains as well as the total weighting value set for each search word, and then output them.



Rule to be created

The rule to be created in the example above is as follows:

```
on inputEventTypeID {
    join("masterDefinitionID", $message = $word)
        output($ID,
               $subject,
               "masterDefinitionID".$word,
               "masterDefinitionID".$weight,
               lookup_count("masterDefinitionID".$word),
               lookup_sum("masterDefinitionID".$weight)) as outputEventTypeID;
}
```

- *inputEventTypeID* is the development asset ID of the event type definition that is the rule target.
- *masterDefinitionID* is the development asset ID of the master definition that corresponds to the master data to be referenced.
- *outputEventTypeID* is the development asset ID of the event type definition (filtered) that represents the results of joining.

5.4.4.2 Complex Event Processing

Define the rules to be used by complex event processing.

Considerations when creating complex event processing rules

The items to consider when creating complex event processing rules are as follows:

- Unit of rule creation

  Create a complex event processing rule for each use application or for each purpose of performing event pattern detection. Decide on a unit of creation in which an event pattern detection described in a certain rule will not affect other rules.

  It is also possible to describe multiple processes in one rule definition, but this will create large rule definitions and may lead to reduced maintainability.

  For example, if events relating to home electronic equipment are to be processed and the content to be detected varies significantly between domestic appliances and information devices, create the following two rule definitions:

  - Rule definition to detect patterns in events relating to domestic appliances

- Rule definition to detect patterns in events relating to information devices

- Referencing external data

  Consider whether referencing external data is necessary in event processing. A Terracotta cache and a relational database (RDB) can be used as external data.

  Refer to "5.4.4.3 Terracotta Collaboration" for information on referencing a Terracotta cache.

  Refer to "5.4.4.4 RDB Collaboration" for information on referencing a relational database.

- Whether processing results are to be sent or logged

  The processing results of complex event processing rules can be sent to an external Web service using SOAP but they can also be processed using a Java class deployed to a CEP engine, or logged in an event log using logging. Apply the SOAP listener, custom listener, and logging listener, respectively, to the complex event processing rules in these cases.

  Refer to "5.4.4.5 SOAP Listener" for information on how to use the SOAP listener.

  Refer to "5.4.4.6 Custom Listener" for information on how to use the custom listener.

  Refer to "5.4.4.7 Logging Listener" for information on how to use the logging listener.

- Rule creation procedure

  Design complex event processing rules in stages, without describing statements from the outset, and develop them so that the intended events will be reliably detected. The creation procedure for complex event processing rules is shown below.

Creation procedure for complex event processing rules

This section uses examples to explain the following creation procedure for complex event processing rules:

1. **Consider what is to be achieved by using complex event processing.**

   Decide on what is to be achieved by using complex event processing. If this is unclear at this point, analyzing the collected events can sometimes clarify this.

   Example

   Example of a rule

   "When someone is home, if rain is likely, recommend using the drying feature of the washing machine."

2. **Consider the events, external data to be referenced, and output content.**

   Consider the events, the external data for referencing, and the output content required to create the rule. Also consider using a named window for retaining events in memory.

   Example

   Examples of events, external data, output, and named window

   The rule example above uses the following events, external data, output content, and named window:

   - Events

     - TV control event
       For the household appliance event data sent from the home gateway of each household, the value of the device category property is to be "Television".

     - Weather forecast event
       The forecast is to be represented by the time and weather.

   - External data

     - Washing machine model information (whether it has a drying feature)
       Whether the washing machine in that household has a drying feature is to be apparent from the home gateway ID.

- Output content
    - Home gateway ID of the household to be given the recommendation
    - Recommendation details
- Named window
    - Named window for weather forecast events
      The weather forecast events for each time are to be retained for one day.

3. **Refine the processing content.**

   Refine the content of the events and of the complex event processing for detecting them.

   Here, "refine" is the task of using events and specific information such as complex event processing conditions and external data to make what was previously expressed in everyday language as "what is to be achieved" into a representation closer to the rule to be created.

Example

Example of refining the content of complex event processing

The table below shows the results of refining the "When someone is home, if rain is likely, recommend using the drying feature of the washing machine" rule.

Element of the rule	Refined processing content
"When someone is home"	Determine that someone is home in the household where the TV was controlled (detect a TV control notifying event).
"If rain is likely"	Reference the weather forecast information stored in the named window and, from the time of the weather forecast and the time of the TV control, check whether or not there is a forecast of rain after this time.
"Drying feature of the washing machine"	Obtain product information on the washing machine connected to the home gateway from the Terracotta cache.
"Recommend"	If there is a forecast of rain and if the washing machine has no drying feature, recommend hanging the clothes inside the house. If there is a forecast of rain and if the washing machine has a drying feature, recommend using it.

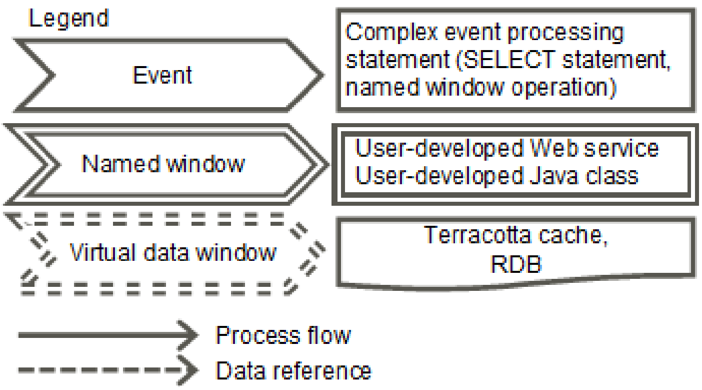
4. **Create an event flowchart.**

   After refining the processing content is completed, summarize the event processing flow to create an event flowchart. An event flowchart associates events and their processing content in chart form. Create an event flowchart before describing the processes using complex event processing rule language, as it is useful for checking the processing content to be achieved.

Information

Legend of event flowchart

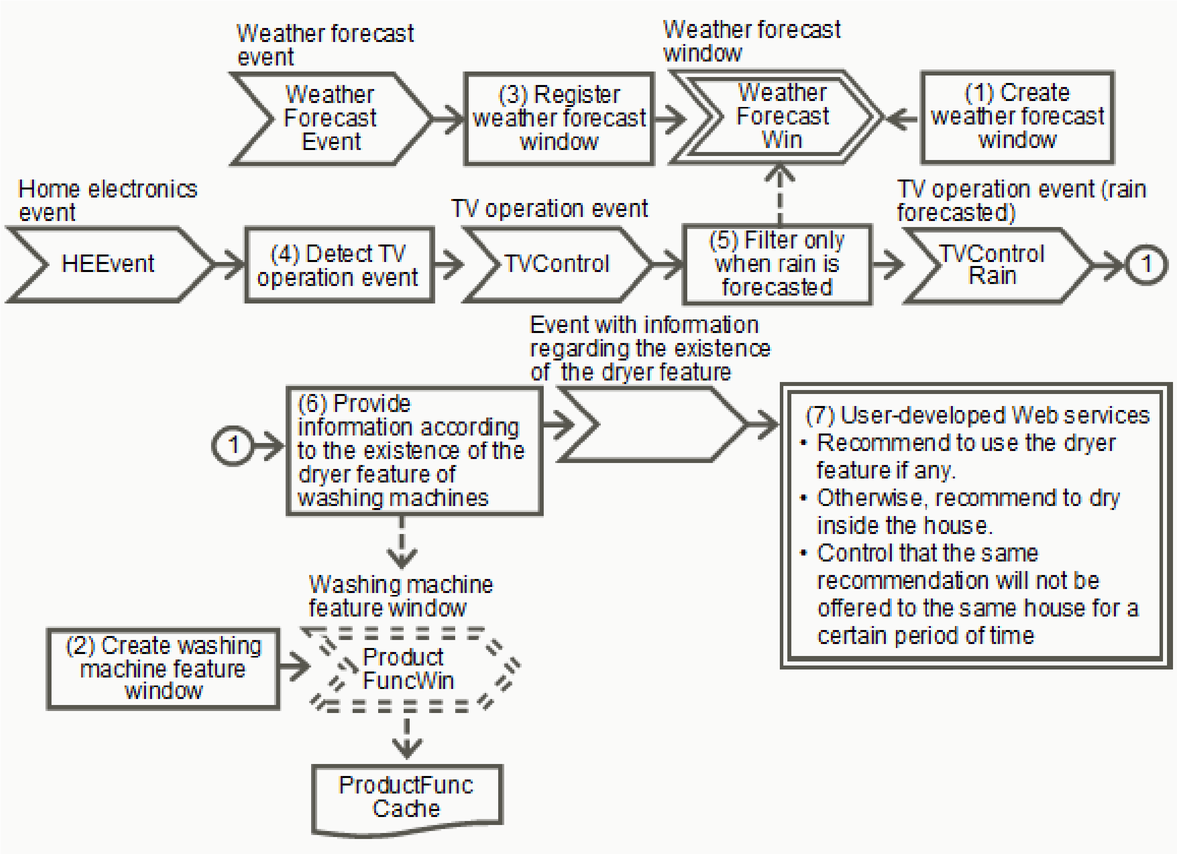
This is the legend for the event flowcharts to be used in this manual.


Legend
Event
Complex event processing statement (SELECT statement, named window operation)
Named window
User-developed Web service
User-developed Java class
Virtual data window
Terracotta cache, RDB
Process flow
Data reference

Example

Example of an event flowchart

This is an event flowchart for "When someone is home, if rain is likely, recommend using the drying feature of the washing machine".


Weather forecast event
Weather Forecast Event
(3) Register weather forecast window
Weather forecast window
Weather Forecast Win
(1) Create weather forecast window
Home electronics event
HEEvent
(4) Detect TV operation event
TV operation event
TVControl
(5) Filter only when rain is forecasted
TV operation event (rain forecasted)
TVControl Rain
1
Event with information regarding the existence of the dryer feature
1
(6) Provide information according to the existence of the dryer feature of washing machines
(7) User-developed Web services
• Recommend to use the dryer feature if any.
• Otherwise, recommend to dry inside the house.
• Control that the same recommendation will not be offered to the same house for a certain period of time
Washing machine feature window
(2) Create washing machine feature window
Product FuncWin
ProductFunc Cache

1. Create a named window for retaining weather forecast events.
2. Create a Virtual Data Window (Terracotta cache) to see whether the washing machine has a drying feature.
3. Store weather forecast events in the weather forecast window.
4. Detect any TV control events from among the household appliance events.
5. Check the weather forecast from after the time that the TV control events occurred and leave only those events with a forecast of rain.

6. Search the washing machine feature window for the households for which a TV control was performed and for which there is a weather forecast of rain, and then add recommendation information according to whether the washing machine has a drying feature and send it to the user-developed Web service.

7. Based on the information received by the user-developed Web service, send a recommendation to the household. Control the user-developed Web service so that the same recommendation is not made to the same household twice within a fixed time period.

5. **Create complex event processing rules.**

Describe complex event processing rules that correspond to the respective elements in the event flowchart.

Example

Example of complex event processing rules that correspond to the event flowchart

Refer to Chapter 1, "Complex Event Processing Language Reference" in the *Developer's Reference* for information on the meanings of rules.

```
// 1. Create a named window for retaining weather forecast events.
@Name('WeatherWin')
create window WeatherForecastWin.std:unique(FORCASTIME).win:time(1 day)
  (FORCASTIME long, WEATHER string);

// 2. Create a Virtual Data Window to check whether the washing machine
//    has a drying feature.
@VDW(cacheName='ProductFuncCache', keyProperty='gatewayId')
create window ProductFuncWin.vdw:ehcache('ProductFuncCache', 'gatewayId')
  as (gatewayId string, dryFunc string);

// 3. Store weather forecast events in the named window.
@Name('InputWeather')
insert into WeatherForecastWin
  select weathfore.FORCASTIME as FORCASTIME, weathfore.WEATHER as WEATHER
    from WeatherForecastEvent as weathfore;

// 4. Detect TV control events.
@Name('ChkTVEvent')
insert into TVControl
  select heevnt.gatewayId as gatewayId, heevnt.updateTime as updateTime
    from HEEvent as heevnt
    where heevnt.deviceCategory = 'Television';

// 5. Check the weather when TV control events occurred (detect forecast
//    of rain).
// Use the following expression to evaluate the time zone of weather
// forecasts. Assume FORCASTIME is "long" (milliseconds):
//   timeOfWeatherForecast
//      <= updateTimeOfTVcontrolEvent
//      <= timeOfWeatherForecast + 1 hour (3600000 milliseconds)
@Name('GetRainyEvent')
insert into TVControlRain
  select tvevnt.gatewayId as gatewayId
    from TVControl as tvevnt unidirectional, WeatherForecastWin as weather
    where updateTimeToMillis(tvevnt.updateTime) between weather.FORCASTIME
      and (weather.FORCASTIME + 3600000)
      and weather.WEATHER = 'rainy';

// 6. Check whether the washing machine has a drying feature and send the
//    recommendation.
// Check whether ProductFuncWin dryFunc property has drying feature ("1": Yes).
// Use the SOAP listener to send the output to the application.
//   'USE_DRY_FUNC' = Recommendation ID if there is drying feature
//   'HANG_LAUNDRY_INSIDE' = Recommendation ID if there is no drying feature
// Use the logging listener to log the output.
//   table      : set log storage area.
//   properties : set property names output (output results)
//                by delimiting with commas.
@Name('PutRecommend')
@SoapListener('soap-001')
@LoggingListener(table='/logsoap',properties='gatewayId,recommendId')
select tvevnt.gatewayId as gatewayId,
  case when product.dryFunc = '1'
       then 'USE_DRY_FUNC'
       else 'HANG_LAUNDRY_INSIDE'end as recommendId
  from TVControlRain as tvevnt unidirectional, ProductFuncWin as product
  where product.gatewayId = tvevnt.gatewayId;
```

5.4.4.3 Terracotta Collaboration

This section explains considerations when performing Terracotta collaboration, and explains how to use Terracotta collaboration, as follows:

- Considerations when using Terracotta collaboration
- Preparing a configuration information file for Terracotta cache
- Using Terracotta cache
  - Creating a Virtual Data Window
  - Using a Virtual Data Window

5.4.4.3.1 Considerations when using Terracotta collaboration

This section explains the items to consider when using Terracotta collaboration for referencing external data in complex event processing, as follows:

- Checking the necessity of Terracotta collaboration

  Unlike when master data is used by the high-speed filter, Terracotta collaboration allows data that is being continually updated to be referenced. (When the high-speed filter is used, master data can be updated using dynamic change, but it cannot be updated continually from a program).

  Data in an external cache can also be added, updated, and deleted when an event occurs.

- Structure of the Terracotta cache

  Data in the Terracotta cache to be referenced by complex event processing rules is managed as entries where each entry is made up of a key and a value. Refer to "5.5.3 Terracotta Cache" for information on the key-value format used for storing in the cache to be used by complex event processing rules.

- Using Terracotta cache

  Use the Virtual Data Window feature to use Terracotta cache with complex event processing. Refer to "5.4.4.3.3 Using Terracotta cache" for information on how to create a Virtual Data Window and how to use a cache via the created Virtual Data Window.

5.4.4.3.2 Preparing a configuration information file for Terracotta cache

To use Virtual Data Window in order to use a Terracotta cache (known as Ehcache), you must place an Ehcache configuration file (ehcache.xml) on the CEP Server. Place the Ehcache configuration file in the following location:

```
/etc/opt/FJSVcep/config/ehcache.xml
```

Refer to the Terracotta manual for information on the Ehcache configuration file. The table below explains the settings required for Terracotta Collaboration.

Element or attribute	Description
ehcache	Root element of the configuration file.
name	Specify the name of the cache manager specified when creating the cache.
maxBytesLocalHeap	Size of the data pool to be used.
terracottaConfig	Element for defining a Terracotta server.
url	List of Terracotta servers in the format "*hostNameOrIpAddress:portNumber*", delimited with a comma (,).
cache	Element for defining cache. Multiple <cache> elements can be specified in a single <ehcache> element.
name	Name of the cache. This is the cache name specified using vdw:ehcache.
terracotta	Defined for using a Terracotta server.
nonstop	Defined for use as nonstop cache.
immediateTimeout	Specify whether to respond with a timeout when a network disconnection is detected. Specify "true" as the value.
timeoutMillis	Specify the standby time until timeout.
timeoutBehavior	Specify operation to be performed if a timeout occurs.
type	Specify "exception" as the value.
searchable	Defined for searching the cache.

Example

The following example uses the cache "`Cache001`" configured on two Terracotta servers (192.168.1.1 and 192.168.1.2) using Terracotta collaboration.

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"
  name="SearchConfig">
  maxBytesLocalHeap="64M">
    <terracottaConfig url="192.168.1.1:9510,192.168.1.2:9510"/>
    <cache name="Cache001">
        <terracotta>
            <nonstop immediateTimeout="true" timeoutMillis="3000">
                <timeoutBehavior type="exception"/>
            </nonstop>
        </terracotta>
        <searchable />
    </cache>
</ehcache>
```

5.4.4.3.3 Using Terracotta cache

This section explains how to create a Virtual Data Window and how to use the created Virtual Data Window.

Creating a Virtual Data Window

Create a Virtual Data Window (hereafter referred to as a "VDW") within complex event processing rules in order to use a Terracotta cache from the rules.

Specifically, describe this as follows:

Syntax: If an event type ID is used

```
create window windowName.vdw:ehcache("cacheName","keyPropertyName") as eventTypeID;
```

Syntax: If type information is specified directly

```
create window windowName.vdw:ehcache("cacheName","keyPropertyName") as (propertyName type,
propertyName type, ...);
```

- Create a VDW using vdw:ehcache() in a CREATE WINDOW statement.

  To reference the cache entity, the *`cacheName`* and the *`keyPropertyName`* for referencing the data must be set. Set them as arguments of vdw:ehcache().

- In *`cacheName`*, specify the name of the cache to be used.
- In *`keyPropertyName`*, specify the property name for identifying the entry. If an event type ID is to be used, specify the property name of the specified event type. If type information is to be specified directly, any name can be specified. The type of the specified property must be a type that corresponds to the "Key" class in the cache.
- In *`windowName`*, specify any name. The name specified here will be used to access the cache from the SELECT statement (INSERT INTO clause, FROM clause), ON SELECT statement, ON UPDATE statement, ON DELETE statement, ON MERGE statement, or subquery in complex event processing rules.
- In *`eventTypeID`*, specify the event specified in the CSV format event type definition. (An XML format event type definition cannot be specified.)
- In *`propertyName type`*, the property name and type that corresponds to the "java.util.HashMap" key to be set in "Value" in the cache must be specified. The property name and its type specified in *`keyPropertyName`* must also be specified.
- If the specified *`propertyName`* has not been set in "java.util.HashMap" to be set in "Value" in the cache, it will be treated as a null by the complex event processing rules.

Example

Definition example for creating a Virtual Data Window (VDW)

This is an example of creating a VDW (`MarketWindow`) to reference a Terracotta cache (`MARKET`).

```
create window MarketWindow.vdw:ehcache("MARKET", "code") as (code string, high int, low int);
```

- "`code`" is specified as a key property.
- "`code` (string type)", "`high` (int type)", and "`low` (int type)" are defined as the properties.

Using a Virtual Data Window

Use a created Virtual Data Window in the same way as an ordinary window. However, to access cache data, use an INSERT INTO clause, join that specifies UNIDIRECTIONAL, ON SELECT statement, ON UPDATE statement, ON DELETE statement, ON MERGE statement, or subquery.

Example

Example of using a created Virtual Data Window (VDW)

This example inserts a `MarketEvent` event into a VDW.

```
insert into MarketWindow select code, high, low from MarketEvent;
```

This example references a VDW `MarketWindow` when a `TicketEvent` event occurs to obtain the data of the VDW events (cache entries) that meet the condition. This example uses `TicketEvent` as a trigger, so UNIDIRECTIONAL is specified in `TicketEvent` for joining the events.

```
select W.high, W.low from TicketEvent as Input unidirectional, MarketWindow as W
    where W.code = Input.code;
```

This example references a VDW (`MarketWindow`) when a `TicketEvent` event occurs to obtain the data of the VDW events (cache entries) that meet the condition. This example uses an ON SELECT statement.

```
on TicketEvent as Input
    select W.high, W.low from MarketWindow as W
        where W.code = Input.code;
```

This example references a VDW (`MarketWindow`) when a `TicketEvent` event occurs to obtain the data of the VDW events (cache entries) that meet the condition. This example uses a subquery.

```
select (select W.high from MarketWindow as W where W.code = Input.code) as high from TicketEvent as
Input;
```

This example updates an event that has the same code in a VDW when a `MarketEvent` event occurs.

```
on MarketEvent as New
    update MarketWindow as W
        set high = New.high, set low = New.low
            where W.code = New.code;
```

Note

Notes on using a Virtual Data Window

- Do not specify another view.

  Do not specify another view such as win:length(1) at the same time as the vdw:ehcache() specification in a CREATE WINDOW statement that defines a Virtual Data Window. Specifying another view does not cause a syntax error, but the view specification cannot be used to operate Terracotta cache data (even if you specify win:length(1), the number of events in the cache will not be "1").

- An INSERT INTO clause is not merely an insertion.

  A Terracotta cache holds only one event (cache entry) for the value of a key property. Therefore, if a new event is inserted into a Virtual Data Window using an INSERT INTO clause and the cache already contains an event that has the same key property value, the event is updated using the new event.

- Cache entries added outside the CEP engine do not propagate.

  For a simple SELECT statement that specifies Virtual Data Window in a FROM clause or a join that does not specify UNIDIRECTIONAL, an event inserted into the Virtual Data Window by the INSERT INTO clause using the complex event processing rules on the same CEP engine propagates. However, an event (cache entry) added from a Terracotta application or a different CEP engine does not propagate. To access cache data, use an ON SELECT statement, a subquery, or a join that specifies UNIDIRECTIONAL.

  The following example shows a simple SELECT statement:

  ```
  select W.high, W.low from MarketWindow;
  ```

  The following example shows a join that does not specify UNIDIRECTIONAL:

  ```
  select W.high, W.low from TicketEvent.std:lastevent() as Input, MarketWindow as W
      where W.code = Input.code;
  ```

  If the INSERT INTO clause inserts an event into `MarketWindow`, the inserted event propagates to the SELECT statements. However, events added outside the CEP engine does not propagate.

- A WHERE clause must uniquely identify cache entries.

  The WHERE clause, which can be used for accessing information stored in a Virtual Data Window, must contain a condition for uniquely identifying cache entries. This condition uses "=" to perform a comparison with the key property specified using vdw:ehcache(). An example is shown below.

In the following example, `W.code = T.code` is valid, because it uniquely identifies a cache entry.

```
create window MarketWindow.vdw:ehcache("MARKET", "code") as (code string, high int, low int);

on TicketEvent as T
    select W.high, W.low from MarketWindow as W
        where W.code = T.code and ( T.price > W.high or T.price < W.low);
```

The table below shows valid and invalid definition examples of using the above rule to change the WHERE clause only.

No.	Definition example	Valid/ Invalid	Explanation
1	`where W.code = '1111'`	Valid	Valid because the condition can uniquely identify a cache entry by using "=" to perform a comparison with the key property
2	`where ( T.price > W.high or T.price < W.low) and W.code = T.code`	Valid	Preceded by a different condition but valid because it includes "=" for performing a comparison with the key property and can uniquely identify a cache entry
3	`where W.high = 1000`	Invalid	Invalid because a comparison using "=" is not performed for the key property.
4	`where W.code > '1111'`	Invalid	Invalid because an operation other than "=" is performed for the key property.
5	`where W.code = '1111' or W.high = 1000`	Invalid	Includes "=" for performing a comparison with the key property but is invalid because there is an OR condition and a cache entry cannot be uniquely identified

5.4.4.4 RDB Collaboration

This section explains the items to consider when implementing RDB collaboration and how to use it.

To reference a relational database as external data, create an RDB reference definition that contains information about connection to the relational database.

This section explains the following items:

- Considerations when using RDB collaboration
- Specifying RDB referencing in complex event processing rules

5.4.4.4.1 Considerations when using RDB collaboration

The items to consider when using RDB collaboration are as follows:

- Creating an RDB collaboration definition

  Using RDB collaboration requires an RDB reference definition in addition to a rule definition. Refer to "5.4.7 Designing an RDB Reference Definition" for information on the RDB reference definition.

- Contents of the relational database

  You must consider the contents of the relational database from the viewpoint of which data, out of the data that cannot be obtained from an input event and the static data that relates to the properties of an event, is also required for checking rules. You can reference RDB data by issuing a complex event processing rule (SELECT statement).

Example

Example of information to be prepared for a relational database

Information that is likely to be required is localities (addresses).

Even if an ID for identifying a customer can be obtained from the contents of an event, the locality (address) of the customer is not always contained in the contents of the event.

In this case, preparing a relational database that registers the relationship between the customer ID and the locality (address) makes it possible to use a customer ID from an event to reference the locality (address) of the customer. The relational database can also be used for processing based on locality (address).

- Notes on using multibyte characters

  Multibyte characters cannot be used for definition names in a relational database such as table names and table item names to be referenced from the complex event processing language. The same applies to other names such as database names, schema names, and user names specified in an RDB reference definition.

  Multibyte characters can be used for item values. In this case, set unicode as the character encoding (or character set) of the relational database. Refer to the manual for the collaboration destination RDB for details.

- Use of a time-limited cache

  When a relational database is referenced for the first time, a query key and the results are stored in a time-limited cache of the CEP engine. Subsequent RDB referencing using the same key entails obtaining the data from the cache. To use a time-limited cache, you must set a cache retention period and a cache purge interval in the RDB reference definition. Refer to "5.4.7 Designing an RDB Reference Definition" for information on the RDB reference definition.

5.4.4.4.2 Specifying RDB referencing in complex event processing rules

You can use the results of a query to a relational database by specifying them using the following syntax in a FROM clause of complex event processing rules.

Syntax:

```
sql:databaseName [" sqlQuery "] or
sql:databaseName [' SqlQuery ']
```

In *databaseName*, specify the "development asset ID" specified in the RDB reference definition.

Enclose *sqlQuery* in double quotation marks (") or single quotation marks ('), and enclose this specification in square brackets "[" and "]".

Alternate parameters can be included in *sqlQuery*. Specify alternate parameters in the ${*expression*} format. The expression is evaluated when the statement is executed.

Note

- Minimize RDB referencing, because it may cause a decline in the performance of Complex Event Processing.
- Multibyte characters cannot be used for definition names in a relational database, such as table names and table item names to be referenced from the complex event processing language. Multibyte characters can be used for item values.
- To specify a nonnumeric literal enclosed in single quotation marks (') in an SQL query that is further enclosed in single quotation marks ('), use the escape notation (\') or the Unicode notation (\u0027).

Example

Example of using RDB referencing

This example uses RDB referencing in a complex event processing rule (SELECT statement).

```
@Name("PutRecommend")
@SoapListener("soap-001")
select tvevnt.gatewayId as gatewayId,
  case when db.DRY_FUNC = '1' then 'USE_DRY_FUNC' else 'HANG_LAUNDRY_INSIDE' end as recommendId
  from TVControlRain as tvevnt,
      sql:app_db[ 'SELECT DRY_FUNC FROM PRODUCTFUNC_TBL WHERE HGW_ID=${tvevnt.gatewayId}' ] as db;
```

- The development asset ID of the RDB reference definition to be used is set to "`app_db`".
- The relational database table being referenced is "`PRODUCTFUNC_TBL`".
- The alternate parameter "`${tvevnt.gatewayId}`" is specified in the condition for referencing the relational database table.
- The alias "`db`" is assigned to the reference results and is used in a SELECT statement.

5.4.4.5 SOAP Listener

Use the SOAP listener to send the processing results of complex event processing statements to a user-developed Web service using SOAP.

To use the SOAP listener, assign the `@SoapListener` annotation in front of the complex event processing statement (SELECT statement) for which the processing results are to be sent.

Refer to "5.4.8 Designing a SOAP Listener Definition" for information on the send destination of processing results and the contents of SOAP messages.

Syntax

```
@SoapListener("soapListenerDefinitionId")
complexEventProcessingStatement(selectStatement)
```

soapListenerDefinitionId

Specify the development asset ID in the SOAP listener definition that defines information such as the URL that is the send destination of the Web service to which the processing results are to be sent.

complexEventProcessingStatement (selectStatement)

Specify the complex event processing statement (SELECT statement) for which the processing results are to be sent.

5.4.4.6 Custom Listener

Use the custom listener to pass the results of complex event processing statements to a Java program (hereafter referred to as a user-developed Java class) for processing.

To use the custom listener, assign the `@CustomListener` annotation in front of the complex event processing statement (SELECT statement) to which you want to pass the processing result.

Refer to "5.6.3 Designing a User-developed Java Class" for information on user-developed Java classes.

Syntax

```
@CustomListener(mainClass="nameOfUserDevelopedJavaClass" [, args={"argument1", "argument2", ...}])
complexEventProcessingStatement(selectStatement)
```

Alternatively, the format may use single quotation marks (') instead of double quotation marks (").

nameOfUserDevelopedJavaClass

Specify the name of the Java class that receives the results of complex event processing statements. Use FQCN format (format that includes the package name) for specifying this name. The CustomListener interface must have been implemented for this Java class.

argument1, argument2, ...

Specify these arguments for the user-developed Java class. If there is no need to pass arguments, omit the "`args`" parameter.

5.4.4.7 Logging Listener

Use the logging listener to log the processing results of complex event processing statements in the event log using Logging.

To use the logging listener, assign the `@LoggingListener` annotation in front of the complex event processing statement (SELECT statement) for which the processing results are to be logged.

Syntax

```
@LoggingListener(table="logStorageArea", properties="propertyNameToBeOutput")
complexEventProcessingStatement(selectStatement)
```

logStorageArea

Use an absolute path to specify the path in the Hadoop system in which the event log is logged.

Even if events are to be logged in the engine log of the CEP Server (if "`file`" is specified in the "`type`" element of the engine configuration file), specify a virtual path name that begins with a slash (`/`) (for example, `/eventName`) to identify the events.

Note

The "`table`" specification is mandatory. Even if a null value is specified, such as '`table=""`', the CEP engine will start normally but logging will not be performed.

propertyNameToBeOutput

Out of the processing results of the SELECT statement, specify the property name to be logged. Multiple properties can also be specified if delimited using commas (`,`).

For a property with nested processing results, use periods (`.`) between the nested properties to join them.

Example

If a "child" property is nested in a "parent" property

If a "child" property is nested in a "parent" property, as shown below, specify the "child" property by describing "parent.child".

```
<root>
    <parent>
        <child>aaa</child>
    </parent>
</root>
```

Note

The "`properties`" specification is mandatory. Even if a null value is specified, such as '`properties=""`', the CEP engine will start normally but logging will not be performed.

complexEventProcessingStatement (selectStatement)

Specify the complex event processing statement (SELECT statement) for which the processing results are to be logged using logging.

Note

- The logging to be defined in an event type definition and the logging to be defined in a rule definition (logging listener) can have separate log storage areas as output destinations. However, note that logging to a different Hadoop system is not possible.
- If the value of the output property is "`null`", this is converted to a blank space before being output to the event log.
- If the output property is a numeric item, this undergoes string conversion before being output to the event log.
- If an output property name that does not exist is specified, a blank space is output to the event log.
- If there are double quotation marks (`"`) in data, these will be output in duplicate within double quotation marks in the data.

  Output example of '`aa"bb"aa`':

```
"aa""bb""aa"
```

5.4.5 Designing a Master Definition

If master data is to be used by high-speed filter rules, design a master definition.

This section explains the items to consider when designing a master definition:

- Timing of updates to the master data

  Updates to the master data will be reflected when the CEP engine is next started.

  Updates will also be reflected when the master data is dynamically changed.

- Amount of memory used by the master data

  The master data is loaded in the CEP engine memory when the CEP engine starts. Refer to "3.3.1.2 Amount of Memory when Master Data is used by the High-speed Filter" for information on the memory required.

5.4.6 Designing an Event Type Definition (Filtered)

If filtered events are to be passed to complex event processing in a format that is different from their original format, due to processes such as join processing with the master data using the high-speed filter, they require an event type definition that corresponds to the format used when they are passed to complex event processing.

This section explains the items to consider when designing an event type definition (filtered):

- Items that make up the filtered events

  Check the items of the events output by high-speed filter rules.

  The number of items specified in the output expression must match the number of items defined in the CSV column information.

- Event format

  The event format can be specified only in CSV format.

5.4.7 Designing an RDB Reference Definition

To reference a relational database as external data, create an RDB reference definition that contains information about connection to the relational database.

This section explains the following items:

- Considerations when Designing an RDB Reference Definition
- Settings for Cache Retention Period and Cache Purge Interval

5.4.7.1 Considerations when Designing an RDB Reference Definition

The items to consider when designing an RDB reference definition include:

- Unit of creation

  Only one RDB reference definition can be deployed per CEP engine.

- Use of a time-limited cache

  When a relational database is referenced for the first time, a query key and the results are stored in a time-limited cache of the CEP engine. Subsequent RDB referencing using the same key entails obtaining the data from the cache. To use a time-limited cache, you must set a cache retention period and a cache purge interval in the RDB reference definition.

  Before using a time-limited cache, you must estimate the memory space required for the cache.

5.4.7.2 Settings for Cache Retention Period and Cache Purge Interval

As the volume of data to be stored in the cache increases, memory usage in the CEP engine increases, so you must carefully consider the values to be specified for the cache.

The cache retention period is the period for which data is to be held in the cache. Specify a value in seconds from 0 to 2147483647.

The cache purge interval is the interval at which the cache is checked for the purpose of flushing the cache if the retention period has elapsed. Specify a value in seconds from 1 to 2147483647.

The concept behind the cache retention period and the cache purge interval is explained below.

If RDB data will not be updated

Setting a high value for the cache retention period poses no problem. Flushing a large volume of data from the cache causes garbage collection, and processing performance is temporarily degraded. Take into account the volume of data to be input to the cache, and set a cache purge interval that is unlikely to cause garbage collection.

If update of RDB data is expected

Even when RDB data is updated, the old cache data may be used for up to the cache retention period + cache purge interval. Set the cache retention period and cache purge interval according to the length of time for which the old data is to be used after the RDB data is updated. Data is held in the cache at each query, so it is possible that old data held in the cache may be obtained for some queries, and new updated data may be obtained for other queries. When designing rules for referencing a relational database, consider whether this situation is possible.

5.4.8 Designing a SOAP Listener Definition

A SOAP listener definition specifies the interface of the user-developed Web service to which the processing results of complex event processing rules are to be sent. This section explains the points to consider in a SOAP listener definition, as follows:

- Unit of creation

  Create a SOAP listener definition for each user-developed Web service. One SOAP listener definition can be used when a generic application is created, and an application can also be separated by pattern matching.

- Development asset ID

  Specify a name for the SOAP listener definition that is unique in the CEP engine in which it will be deployed and that suggests the processing content called when rule matching finds a match.

  A simple example is where a name such as *powerOn* is used for a process to notify when the power is on.

- Association between a SOAP listener definition and a user-developed Web service

  Check that the property value of the corresponding complex event processing rule matches the user-developed Web service to be called, based on the WSDL (interface definition) of the application. Even if an existing Web service is to be used, consider factors such as whether the interface can be used as it is or if it needs to be changed. The aspects that must match are as follows:

  - Parameter names

    The parameter names to be passed to the user-developed Web service must match the property names selected from within the complex event processing rules.

    If the property names selected from within the complex event processing rules are to be used as the parameter names to be passed to the user-developed Web service, use them by defining aliases within the rules.

  - Parameter types

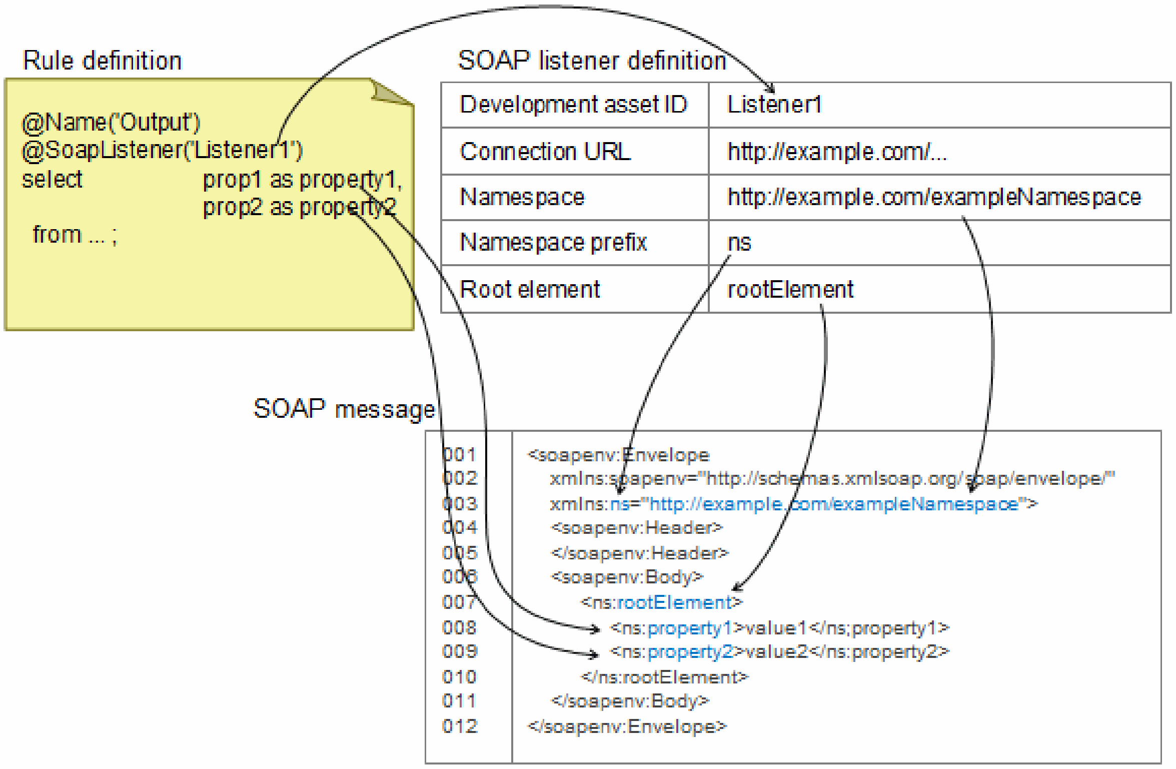
    The parameter types must also match the types used from within the complex event processing rules.

  - Parameter order

    The order of parameters in a SOAP message output by the SOAP listener will not necessarily be the order of properties specified in the SELECT statement. If a user-developed Web service is to check the order of the parameters, you must first check the SOAP message output by the SOAP listener and ensure that the order is the same.

Below is an example of the association between a rule definition and a SOAP listener definition. The SOAP messages to be sent to a user-developed Web service are generated from the rule definition and from the SOAP listener definition associated with it.

Figure 5.4 Example of the association between a rule definition and a listener definition, and the SOAP messages to be sent



Rule definition

```
@Name('Output')
@SoapListener('Listener1')
select          prop1 as property1,
                prop2 as property2
 from ... ;
```

SOAP listener definition

Development asset ID	Listener1
Connection URL	http://example.com/...
Namespace	http://example.com/exampleNamespace
Namespace prefix	ns
Root element	rootElement

SOAP message

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://example.com/exampleNamespace">
  <soapenv:Header>
  </soapenv:Header>
  <soapenv:Body>
    <ns:rootElement>
      <ns:property1>value1</ns;property1>
      <ns:property2>value2</ns:property2>
    </ns:rootElement>
  </soapenv:Body>
</soapenv:Envelope>
```

5.5 Design (Data)

This section explains how to design each type of data.

5.5.1 Event Data (for Testing)

Consider using event data sent from an event sender application to check the operation of created rules.

Pay attention to the following points when considering this:

- Testing scenario (content of event data required)

  Provide event data based on the scenario envisaged for the rules being designed and developed.

  Provide data by envisaging that abnormal data as well as normal data will be sent.

- Format of event data

  Provide event data using a format that suits the event sender application to be used.

  Provide data in CSV format if the event sender sample program included in the samples for BDCEP is to be used as an event sender application for testing.

5.5.2 Master Data (for the High-speed Filter)

Pay attention to the following points when considering the use of master data by the high-speed filter:

- Format of the master data

  Refer to "5.5.2.1 Format of Master Data".

5.5.2.1 Format of Master Data

The master data consists of data files in CSV format and a schema file.

Refer to "9.7 CSV Format Supported" for information on the CSV format.

The character code that can be used is UTF-8.

Specify LF or CRLF as the newline code to be described at the end of records.

Schema file

This is a file in which only lines of item names are described.

If a schema file contains information other than item names, an error will occur.

Data file

This is a file in which data is stored. If the item names are described in the first line of the data file, skipping the first line can be set in the master definition. Refer to "9.2.3 Master Definition File" for details.

Example

Example of a schema file

```
"Kbn","Number","Code","Name","Value","Total","Comment"
```

Example of a data file

```
"01","1001","AAA","BlockA","1,000","1,000","Comment: Memo number 4023"
"02","1001","BBB","BlockB","","1,200","Comment: Memo number 4023"
"03","1002","CCC","BlockC","800","800","Comment: Memo number 4023"
```

5.5.3 Terracotta Cache

Pay attention to the following points when considering using Terracotta collaboration to reference an external Terracotta cache from complex event processing rules:

- Initial data of the cache

  Initial data must be provided for the cache as required.

  Select a data format according to the specifications of the Terracotta application for update to be used.

- Format of the Terracotta cache

  A Terracotta cache must follow a prescribed format.

  If an existing cache is to be used, check that it is consistent with "5.5.3.1 Terracotta Cache Compatible Formats".

5.5.3.1 Terracotta Cache Compatible Formats

A Terracotta cache consists of key-value pairs. The cache referenced by BDCEP must have the following configuration:

Type used for the key	Type used for the value
java.lang.String	java.util.HashMap<java.lang.String, java.lang.Object>

Each property to be specified in a complex event processing rule corresponds to each "HashMap" element above.

The table below shows the type that is compatible for the value of each "HashMap" element and the corresponding type in complex event processing rules.

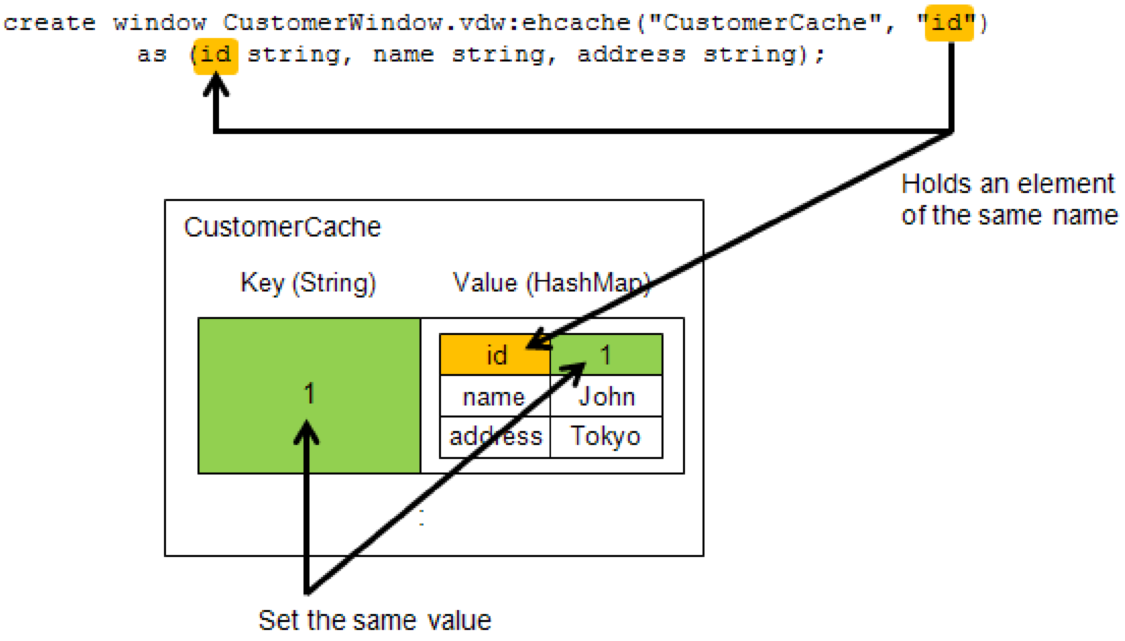
Type used in each "HashMap" element	Corresponding type in complex event processing rules
java.lang.String	string

Type used in each "HashMap" element	Corresponding type in complex event processing rules
java.lang.Character	char/character
java.lang.Boolean	bool/boolean
java.lang.Byte	byte
java.lang.Short	short
java.lang.Integer	int/integer
java.lang.Long	long
java.lang.Float	float
java.lang.Double	double

The Terracotta cache key and "HashMap" to be specified for the value must have the following relationship:

- Ensure that the name of the property to be specified as the cache key is the same as the key of the corresponding "HashMap" element.
- Set the value of the above "HashMap" element as the value of the cache key.

The following figure illustrates this relationship.



```
create window CustomerWindow.vdw:ehcache("CustomerCache", "id")
          as (id string, name string, address string);
```

Refer to "5.6.5 Designing a Terracotta Application" for information on the actual method of use.

5.5.4 Relational Database (RDB)

If referencing an external relational database from the complex event processing rules using RDB collaboration, consider the following points:

- Notes on using multibyte characters

  Multibyte characters cannot be used for definition names in a relational database such as table names and table item names to be referenced from the complex event processing language. The same applies to other names such as database names, schema names, and user names specified in an RDB reference definition.

  Multibyte characters can be used for item values. In this case, set unicode as the character encoding (or character set) of the relational database. Refer to the manual for the collaboration destination RDB for details.

- Characters and maximum length of user names (or access IDs in an RDB reference definition)

  Depending on the type of relational database to be used, there may be restrictions on user names (access IDs).

  For example, if using Symfoware Server with the native interface, specify the user name (access ID) to be set when connecting from the CEP Server in up to 36 alphanumeric characters starting with an alphabetic character. You cannot use a period (.) or multibyte characters.

  Similarly, there are restrictions on the characters and length of database names, schema names, database URLs, and access passwords. Refer to "9.2.4 RDB Reference Definition File" for information on the characters and length of each item.

- Format of relational database tables

  The tables of the relational database to be used must conform to the prescribed format.

  If using an existing relational database, ensure that it matches "5.5.4.1 Supported RDB Table Formats".

- Initial data of a relational database

  If preparing a new relational database, you must prepare initial data.

  You need not prepare initial data if using an existing relational database.

  The format of initial data must conform to the specifications for the RDB commands to be used for storing data. Refer to the manual for the collaboration destination RDB for details.

5.5.4.1 Supported RDB Table Formats

Enter the table names and table item names of the relational database in an alphanumeric string containing up to 36 characters and starting with an alphabetic character.

The table below lists the types that can be used in each item of a relational database table. Refer to the manual for the collaboration destination RDB for information on each type.

Table 5.7 Types that can be used in RDBs

CHAR	VARCHAR	NCHAR	NCHAR VARYING
NUMERIC	DECIMAL	INTEGER	SMALLINT
FLOAT	REAL	DOUBLE PRECISION	DATE
TIME	TIMESTAMP	BLOB	

Note

Multibyte characters cannot be used for definition names in a relational database such as table names and table item names to be referenced from the complex event processing language. Multibyte characters can be used for item values.

5.6 Design (Collaboration Application)

This section explains how to design each type of collaboration application.

5.6.1 Designing an Event Sender Application

Consider which method to use to send events to the CEP engine.

Here, consideration must be given to which communication method to use for the input events and which application to use to send the events, according to that communication method.

Communication method for input events

Consider which communication method to use when the CEP engine is receiving events, according to the characteristics of the input events.

If an existing system that is to be used as the event issuer has an event sending feature, check which communication methods the existing system can use.

With BDCEP, the following three communication methods can be selected:

- SOAP
- HTTP
- Socket

Select a communication method according to the characteristics of the system or device that is to be used as the event sender, and the desired processing performance. Characteristics of each communication method are described below.

Communication method	Characteristic
SOAP	A generic communication protocol. This method allows sending XML or CSV event data to the CEP engine as SOAP messages. WSDL can be used to define an interface and an event sender application can be developed using the existing Web service development tool.
HTTP	A generic communication protocol. This method allows sending XML or CSV event data to the CEP engine connected via HTTP without modification. There is no unnecessary header information so that the communication load can be less than that of SOAP. This means that this method is not only more suitable for sending large amounts of event data but also enables sending event data from devices supporting the REST communication.
Socket	A communication protocol unique to BDCEP, which is used for TCP/IP socket communication. This method allows simultaneously sending multiple XML or CSV event data. There is no HTTP header so the communication load can be less than that of HTTP, making this method most suitable for sending large amounts of event data.

Event sender application

Consider which system or application to use to send events to the CEP Server.

- There are two types of event sender applications: systems that send events during normal business operation and those that send events at any time during rule testing. For the latter, consider using the event sender sample program supplied with BDCEP, if the volumes of event data are small. Refer to "5.11.6 Event Sender Sample Program" for details.
- If the system issuing the events can send the events using a communication method supported by BDCEP, check that the CEP engine can receive events using that feature.
- If the event issuer has no event sending feature, or if it does not support a communication method of the BDCEP input adapter, consider developing an application to send events to the CEP Server (event sender application).
- The design and development of an event sender application must be carried out according to the specifications of the server or device that is to be used as the event sender. Refer to Chapter 3, "Input Adapter Reference" in the *Developer's Reference* for detailed specifications on each communication method and for information on the sample event sender application.

5.6.2 Designing a User-developed Web Service

Refer to the application server manuals for information on designing and developing a user-developed Web service. This section explains the points to consider when designing a user-developed Web service, as follows.

- Integrating similar processes

  If multiple recommendations are to be made to an individual, rather than separating Web services by recommendation, group them into one Web service to create one SOAP listener definition, and then use calling parameters to divide up the Web service processes to be executed.

- Separating processes with different responses or targets

  If processes have different responses or targets, as with "recommend to a person" and "control a device", the internal logic of the Web service and testing methods will differ, so consider developing such processes as separate applications.

- Execution environment of a Web service with operation checked

  BDCEP checks the operation of user-developed Web services in the Web service execution environments below. If executing using another product, perform sufficient connection testing.

Product name	Details
Interstage Application Server	Checked if a Java EE Web service is used.
Apache Axis2	Open source Web service execution framework. The URL is as follows: http://axis.apache.org/axis2/java/core/

  If a product other than those above is to be used, check that "`Content-Length`", "`Content-Type`", and each value have been set correctly in the HTTP header that will form the response from the user-developed Web service to the CEP engine.

5.6.3 Designing a User-developed Java Class

The custom listener passes the results of complex event processing to a user-developed Java class. This section provides an overview of the user-developed Java classes and the considerations required when designing it.

Implement the following interface for a user-developed Java class:

```
com.fujitsu.cspf.cep.CustomListener
```

The following two methods must be implemented using this interface:

Type of the return value	Method name	Argument	Explanation
void	setArgs	String[] args	The args parameter specified using the `@CustomListener` annotation is passed.
		String statementName	The name (specified using `@Name`) of the complex event processing statement to which the `@CustomListener` annotation was attached is passed.
void	update	Map[] newEvents	The processing result (output event) of the complex event processing statement to which the `@CustomListener` annotation was attached is passed as an array of the java.util.Map object. The output event passed as a java.util.Map object contains a property name and value pair. If the rules output events periodically, the update method may be called even if there is no output event.

The following processes are performed for the user-developed Java class each time there is an output from a complex event processing statement:

1. An instance of the user-developed Java class is generated.
2. The setArgs method is called.
3. The update method is called.

In addition, an instance of a user-developed Java class is generated when the CEP engine is started.

If an exception occurs during custom listener processing, the CEP engine catches the exception and outputs it to the engine log and the system log. Processing of other events continues.

Note

A user-developed Java class runs on the same Java VM as the CEP engine. Take the following points into account when designing a user-developed Java class.

- Design a user-developed Java class so there is no bottleneck in processing time

  Design a user-developed Java class so that it takes only a short time to generate an instance of the user-developed Java class and to call the setArgs and update methods. If these processes take time, events awaiting processing may accumulate in the CEP engine and adversely affect the processing performance of the entire CEP engine. Particularly if a large volume of events is to be output, the impact will be greater.

- Create a thread-safe design

  Processing of a user-developed Java class is called from multiple threads, so create a thread-safe design. For example, if using class variables, you must consider the fact that the user-developed Java class will be called from multiple threads.

  However, an instance of a user-developed Java class is generated each time an output event occurs, and one instance runs on only one thread. Therefore, if processing involves merely operating the instance variables, there is no need to consider multiple threads.

- Throw errors that must be monitored as exceptions

  To monitor errors that occur in a user-developed Java class, throw details of the error as an exception outside the user-developed Java class. An exception that is thrown is caught by the CEP engine and output to the engine log and the system log.

Example

Sample source code of a user-developed Java class is stored in the following directory:

```
/opt/FJSVcep/sample/CustomListener
```

5.6.4 Designing an Event Log Analysis Application

If logging is to be used to accumulate event logs in a Hadoop system, design an application to analyze the content of the accumulated event logs. The application will use the Hadoop API and operate on the Hadoop system.

Refer to the Interstage Big Data Parallel Processing Server (hereafter, referred to as "BDPP") manuals for information on designing and developing applications to operate on a Hadoop system.

The data formats of the event logs to be analyzed by this application are shown below.

5.6.4.1 Output Destination and File Format of an Event Log

Event logs are output to a log storage area specified in the event type definition or in the logging listener in a complex event processing statement. The log storage area that will be the output destination is generated automatically.

If the output destination is a Hadoop system, the details are as follows:

Output destination

The output destination can be changed using the value specified in the `directory` element of the engine configuration file.

If a directory name is specified in the `directory` element, the output destination will be a path made by joining the following values:

- Value set in "`pdfs.fs.local.basedir`" (*1)
- Directory name specified in the engine configuration file
- Log storage area specified in the event type definition or logging listener
- Automatically generated log file name

*1: "`pdfs.fs.local.basedir`" is the Hadoop mount directory. Refer to the BDPP manuals for details.

If a slash (`/`) only is specified in the `directory` element, the output destination will be a path made by joining the following values:

- Value set in "`pdfs.fs.local.basedir`"
- Log storage area specified in the event type definition or logging listener
- Automatically generated log file name

Example

Example of output destination

The output destination will be "`/mnt/pdfs/hadoop/tmp/`*`logFileName`*"for the following conditions:

- If the value set in "`pdfs.fs.local.basedir`" is "`/mnt/pdfs`"; and
- If "`hadoop`" is specified as the directory name in the engine configuration file; and
- If "`/tmp`" is specified as the log storage area specified in the event type definition or logging listener of the complex event processing statement

The output destination will be "`/mnt/pdfs/tmp/`*`logFileName`*" for the following conditions:

- If the value set in "`pdfs.fs.local.basedir`" is "`/mnt/pdfs`"; and
- If a slash (/) is specified as the directory name in the engine configuration file; and
- If "`/tmp`" is specified as the log storage area specified in the event type definition or logging listener of the complex event processing statement

Note

If the output destination of the event log is duplicated and the format of the event data is the same, event data of a different event type will be output to the same file. If analysis is to be performed by event type or by output by logging listener, separate the output destinations.

Log file format

The format will be Hadoop SequenceFile (binary file) format.

Log file name

A log file will be automatically generated in the log storage area using the file name shown below.

This file will be renamed with the "`.done`" extension in 300 seconds by default.

```
dateTime_vmName_branchNumber
```

- *`dateTime`*: yyyyMMddHHmmssSSS
- `vmName`: *processID@cepServerHostName*
- *`branchNumber`*: 0000000001 to 0000000122

Point

A file with the "`.done`" extension will be analyzed by the event log analysis application. Move it to an arbitrary directory to analyze it.

Note

A file with an extension other than "`.done`" is a file that is being output, so do not perform an operation on it.

Upper limit of file size

The upper limit of the file size is LONG MAX (2^{63} - 1).

Upper limit of number of files

None

Key of SequenceFile

The date and time information (yyyyMMddHHmmss) will be the key. The corresponding Hadoop type (API) is "org.apache.hadoop.io.Text".

The date and time above will be the date and time at which the event data was written. (This may differ from the date and time at which the CEP engine received the events.)

Value of SequenceFile

Input events are output as they are. The corresponding Hadoop type (API) is "org.apache.hadoop.io.BytesWritable".

Compression format of SequenceFile

Record compression

Versions of SequenceFile

6

Information

If outputting to the engine log

Input events are output to the engine log unchanged.

5.6.5 Designing a Terracotta Application

If a Terracotta cache is to be referenced in complex event processing rules, consideration must be given to an application for storing initial data in the Terracotta cache. If an existing cache can be used, use an existing Terracotta application.

Refer to the Interstage Terracotta BigMemory Max manuals for information on designing and developing a Terracotta application.

Refer to "5.5.3.1 Terracotta Cache Compatible Formats" for information on the format of the cache to be updated.

Each "HashMap" to be specified in a key and value for a Terracotta cache must be associated as follows:

- Ensure that the name of the property to be specified as the cache key is the same as the key of the element corresponding to HashMap.
- In the key value in the cache, set the value as that of the "HashMap" element above.

Example

Example of a Virtual Data Window

```
create window windowName.vdw:ehcache("CacheA", "key") as (key string, address string);
```

Example of a program to add entries to a Terracotta cache

```
import net.sf.ehcache.*;
import java.util.HashMap;
(...)
CacheManager cacheManager
   = CacheManager.newInstance(Crud.class.getResource("/xxx/ehcache.xml")); ... 1.
Ehcache cache = cacheManager.getCache("CacheA");  ... 1.
String keyProperty = "key";          ... 2.
String keyValue = "1";               ... 2.
HashMap value = new HashMap();       ... 3.
value.put(keyProperty, keyValue);    ... 3.
value.put("address", "Boston");      ... 3.
cache.put(new Element(keyValue, value));  ... 4.
(...)
```

1. Specify the configuration file of the Terracotta cache (Ehcache) to obtain CacheManager, and obtain the cache to be used by the Virtual Data Window.

2. Create the key property name and its value to be specified in "`keyProperty`" of the Virtual Data Window. Here, "`key`" is set as the "`keyProperty`".

3. Create the "java.util.HashMap" object to be stored as the value of the cache. Here, a "java.lang.String" type value is set in the "`address`" property. Also, ensure that the key property name and its value created in "2." are set in the "HashMap" element.

4. Add entries to the cache.

5.7 Implementation

This section explains the implementation tasks (such as coding) for development assets.

5.7.1 Creating a Definition File

This section explains how to create a definition file, as follows:

- Creating an Event Type Definition File
- Creating a Rule Definition File
  - Debug log listener
- Creating a Master Definition File
- Creating an RDB Reference Definition File
- Creating a SOAP Listener Definition File

5.7.1.1 Creating an Event Type Definition File

Create an event type definition file similar to the items of the event type definition designed previously.

Refer to "9.2.1 Event Type Definition File" for information on the format of an event type definition file.

Example

Example of an event type definition

Below is an example of an event type definition for an event in XML format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventType xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="EVENTTYPE_01">
  <comment>Event type definition 01</comment>
  <type>XML</type>
  <xmlSchema>
    <![CDATA[
     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns="http://dataaccesscontrol.sspf.fujitsu.com/namespace/xmlmessage"
       targetNamespace="http://dataaccesscontrol.sspf.fujitsu.com/namespace/xmlmessage">
        <xs:element name="messagedata">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="memberID" type="xs:string" />
              <xs:element name="areaID" type="xs:string" />
              <xs:element name="status" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
     </xs:schema>
    ]]>
  </xmlSchema>
  <root>messagedata</root>
  <useLogging>true</useLogging>
  <loggingTableName>/echonet</loggingTableName>
  <useCep>true</useCep>
</eventType>
```

5.7.1.2 Creating a Rule Definition File

Create a rule definition file similar to the items of the rule definition designed previously.

Refer to "9.2.2 Rule Definition File" for information on the format of a rule definition file.

Also assign a debug log listener (`@DebugLogListener`) in complex event processing rules, to check operation in an integration test. Refer to "5.7.1.2.1 Debug log listener" for information on the debug log listener.

Example

Example of a rule definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rule xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RULE_01">
    <comment>Rule definition 01</comment>
    <filter>
        <![CDATA[
            on EVENTTYPE_01 {
              if ($status == 'Walking') then output() as EVENTTYPE_01;
            }
        ]]>
    </filter>
    <statements>
        <![CDATA[
            @SoapListener('LISTEN_01')
            @DebugLogListener
            select * from EVENTTYPE_01 where areaID = '1010';
        ]]>
    </statements>
</rule>
```

This example describes the following rules:

High-speed filter rule

If the status item contents (string) extracted from the "EVENTTYPE_01" event type input event is "Walking", it is transferred to Complex Event Processing.

Complex event processing rule

This rule notifies the SOAP listener of event data with the "EVENTTYPE_01" event type, and simultaneously outputs debug information to the engine log.

5.7.1.2.1 Debug log listener

Using the debug log listener allows logs for debugging to be output to the engine log when a complex event processing statement is executed.

Specifically, assign the `@DebugLogListener` annotation in front of the target complex event processing statement.

Also, using the `@Name` annotation to give a name to the target complex event processing statement at the same time will allow the output information of the engine log to be found easily.

Syntax

```
@Name("name")
@DebugLogListener
complexEventProcessingStatement
```

name

This is output at the same time as debug log output and allows the output information to be found easily.

If the same name is given to multiple complex event processing statements, names will be automatically assigned using a format of two hyphens (--) and a numeric will be appended to the end of each name.

complexEventProcessingStatement

This is the complex event processing statement to be the target of debug log output.

Note

The debug log listener may cause a decline in performance, so avoid use during normal business operation.

Information

If the `@Name` annotation is not specified

If the `@Name` annotation is not specified, an automatically assigned unique name such as "`0b0562a2-56e7-4cf3-a520-cb1e16ef2992`" will be output in place of the value of the `@Name` annotation.

Example of output when the `@Name` annotation is specified (`@Name('EPL')` specified)

```
2012-07-09 19:32:35,495 [DEBUG] EPL:length=1
EPL[0]
```

Example of output when the `@Name` annotation is not specified

```
2012-07-09 19:35:55,244 [DEBUG] 34b1785f-900c-4420-b2bf-ea53aa368b07:length=1
34b1785f-900c-4420-b2bf-ea53aa368b07[0]
```

5.7.1.3 Creating a Master Definition File

Create a master definition file similar to the items of the master definition designed previously.

Refer to "9.2.3 Master Definition File" for information on the format of a master definition file.

Example

Example of a master definition

This is an example of a master definition (development asset ID: `MASTER_01`) where the schema file is "`/var/tmp/SchemaFile01.csv`" and the data file is "`/var/tmp/MasterFile01.csv`".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<master xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="MASTER_01">
    <comment>Master definition 01</comment>
    <schemaFile>/var/tmp/SchemaFile01.csv</schemaFile>
    <dataFile>/var/tmp/MasterFile01.csv</dataFile>
    <skipHeader>false</skipHeader>
</master>
```

5.7.1.4 Creating an RDB Reference Definition File

Create an RDB reference definition file in accordance with the items of the RDB reference definition you designed.

Refer to "9.2.4 RDB Reference Definition File" for information on the format of the RDB reference definition file.

Example

Definition example for referencing Symfoware (Open Interface)

This example connects to a Symfoware (Open Interface) RDB server (host name: `RERDB001`, port number: `20001`) using the database name "`dbms1`" and user name "`user01`". It also caches the results referenced from the RDB server for 1 minute, and every 2 minutes flushes a cache for which the retention period has elapsed.

Refer to "9.2.4 RDB Reference Definition File" for definition examples referencing other relational databases.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<database xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RDBREF_01">
    <comment>RDB reference definition_01</comment>
    <jdbcClass>org.postgresql.Driver</jdbcClass>
    <url>jdbc:postgresql://RERDB001:20001/mydb?loginTimeout=20</url>
    <user>user01</user>
    <password>pass123</password>
    <maxAge>60</maxAge>
    <purgeInterval>120</purgeInterval>
</database>
```

5.7.1.5 Creating a SOAP Listener Definition File

Create a SOAP listener definition file similar to the items of the SOAP listener definition designed previously.

Refer to "9.2.5 SOAP Listener Definition File" for information on the format of a SOAP listener definition file.

Example

Example of a SOAP listener definition

This example defines the notification of a message (event) that includes a SOAP body saying that the root element is "`cep`" in the user-developed Web service with the connection destination URL "`http://192.168.11.249/WebServWAR/MyApp1Service`".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<soapListener xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="LISTEN_01">
    <comment>SOAP listener definition 01</comment>
    <url>http://192.168.11.249/WebServWAR/MyApp1Service</url>
    <nameSpace>http://webservice/</nameSpace>
    <prefix>ns</prefix>
    <method>cep</method>
</soapListener>
```

5.7.2 Preparing Data

This section explains how to prepare the data to be referenced from rules.

5.7.2.1 Preparing Event Data (for Testing)

Prepare (create) event data to be used in checking the operation of rules.

Use an event data format that suits the event sender application to be used.

Example

This is an example of event data for the event sender sample program supplied with the samples of BDCEP.

/opt/FJSVcep/sample/sample1/event/CouponEvent.csv

```
"STR0001","CPN0001","30"
```

5.7.2.2 Preparing Master Data (for the High-speed Filter)

Prepare (create) a schema file and data files of the master data for the high-speed filter.

Refer to "9.2.3 Master Definition File" for information on the schema file and data files.

Example

Example of a schema file

```
"Kbn","Number","Code","Name","Value","Total","Comment"
```

Example of a data file

```
"01","1001","AAA","BlockA","1,000","1,000","Comment: Memo number 4023"
"02","1001","BBB","BlockB","","1,200","Comment: Memo number 4023"
"03","1002","CCC","BlockC","800","800","Comment: Memo number 4023"
```

5.7.2.3 Preparing Data to be Stored in a Terracotta Cache

If required, prepare (create) the data that is to be initially stored in the Terracotta cache to be referenced from complex event processing rules.

If an existing cache is to be used, this data need not be prepared (created).

Select a data format according to the specifications of the Terracotta application for update to be used.

5.7.2.4 Preparing a relational database

Prepare (create) a relational database to be referenced from the complex event processing rule, and the data to be stored first. You need not prepare (create) data if using an existing relational database.

Create an RDB schema definition and storage data according to the specifications for the RDB commands to be used for storing data. Refer to the manual for the collaboration destination RDB for details.

5.7.3 Implementing a Collaboration Application

This section explains how to implement a collaboration application, as follows:

- Implementing an Event Sender Application
- Implementing a User-developed Web Service
- Implementing a User-developed Java Class (Custom Listener)
- Implementing an Event Log Analysis Application
- Implementing a Terracotta Application

5.7.3.1 Implementing an Event Sender Application

Implement an application to send events to the CEP engine.

Implement the application according to the type of input adapter to be used.

Refer to Chapter 3, "Input Adapter Reference" in the *Developer's Reference* for information on sample programs by input adapter type.

5.7.3.2 Implementing a User-developed Web Service

Implement a Web service application to be called from the SOAP listener.

5.7.3.2.1 Web service implementation procedure

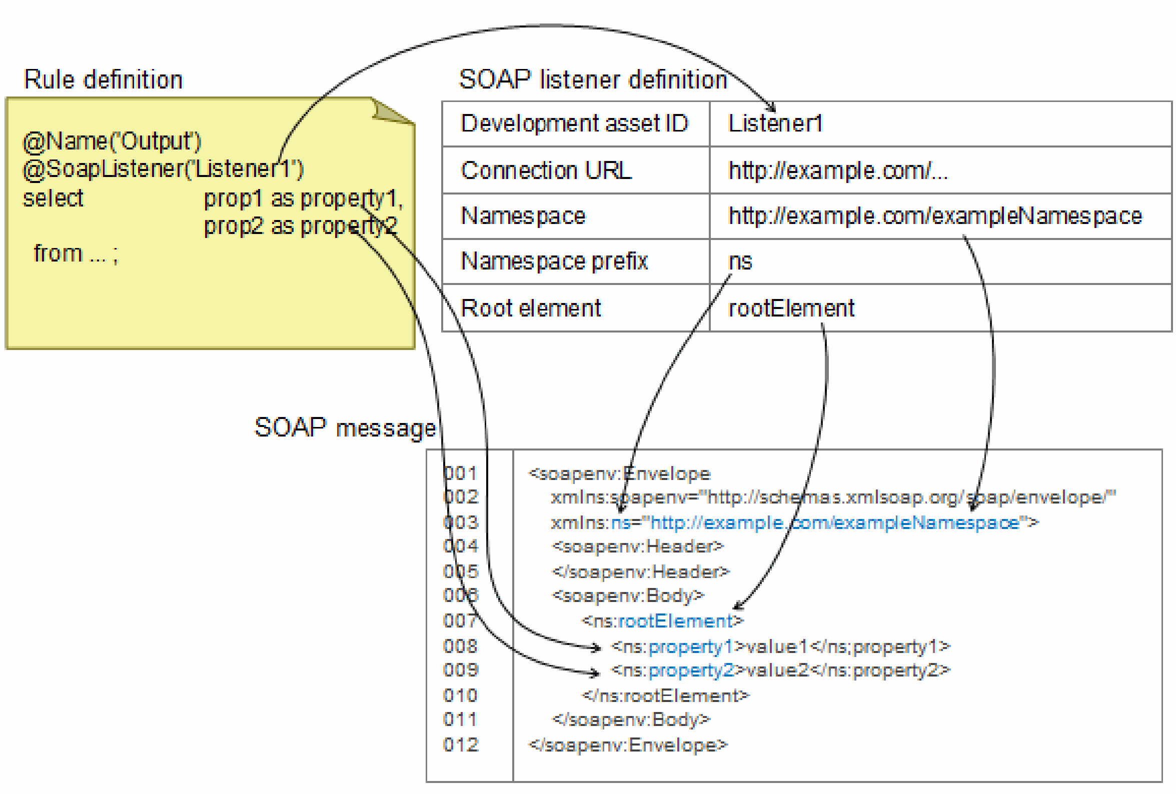
This section explains the Web service implementation procedure.

1. **Create a WSDL.**

   Create a WSDL (interface definition) for a user-developed Web service from the interface information (Web service URL, namespace, prefix, and method) of the Web service to be called, that was defined in the SOAP listener definition, and from the parameters that are specified in the complex event processing statements (SELECT statements).

   Below is an example of the association between the complex event processing statement (SELECT statement) for detecting the target events, the SOAP listener definition associated with that rule, and SOAP messages generated from the SOAP listener.

Figure 5.5 Association between a rule definition, listener definition, and SOAP messages to be sent



```
@Name('Output')
@SoapListener('Listener1')
select          prop1 as property1,
                prop2 as property2
  from ... ;
```

Development asset ID	Listener1
Connection URL	http://example.com/...
Namespace	http://example.com/exampleNamespace
Namespace prefix	ns
Root element	rootElement

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://example.com/exampleNamespace">
   <soapenv:Header>
   </soapenv:Header>
   <soapenv:Body>
      <ns:rootElement>
         <ns:property1>value1</ns;property1>
         <ns:property2>value2</ns:property2>
      </ns:rootElement>
   </soapenv:Body>
</soapenv:Envelope>
```

A sample WSDL for a Web service for receiving these SOAP messages is shown below. This WSDL defines a message receive-only (one-way) Web service. If it is implemented as a Web service that returns a response to the CEP engine (request-response), the CEP engine ignores this response.

Table 5.8 Sample WSDL

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions
  targetNamespace="http://example.com/exampleNamespace"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/exampleNamespace"
  xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <types>
    <xs:schema elementFormDefault="qualified"
```

```
011        targetNamespace="http://example.com/exampleNamespace" >
012        <xs:element name="rootElement">
013          <xs:complexType>
014            <xs:sequence>
015              <xs:element name="property1" type="xs:string" />
016              <xs:element name="property2" type="xs:string" />
017
018            </xs:sequence>
019          </xs:complexType>
020        </xs:element>
021      </xs:schema>
022    </types>
023
024    <message name="notifyMessage">
025      <part name="body" element="tns:rootElement" />
026    </message>
027
028    <portType name="eventReceiverPortType">
029      <operation name="notifyOperation">
030        <input message="tns:notifyMessage" />
031      </operation>
032    </portType>
033
034    <binding name="eventReceiverSOAPBinding" type="tns:eventReceiverPortType">
035      <soapbind:binding transport="http://schemas.xmlsoap.org/soap/http"
036        style="document" />
037      <operation name="notifyOperation">
038        <soapbind:operation soapAction="" />
039        <input>
040          <soapbind:body use="literal" />
041        </input>
042      </operation>
043    </binding>
044
045    <service name="eventReceiverService">
046     <port name="eventReceiverSOAPPort" binding="tns:eventReceiverSOAPBinding">
047        <soapbind:address location="http://example.com/serviceEndPoint" />
048      </port>
049    </service>
050  </definitions>
```

Modifying, as follows, the underlined parts according to a rule definition or SOAP listener definition allows the WSDL above to be used to generate a template of the Web service to be created:

- Line numbers 003, 005, and 011

  Use the value of the namespace in the SOAP listener definition as the target namespace of the WSDL ("targetNamespace" attribute of the "definitions" element), the declaration of the target namespace prefix ("xmlns:tns" attribute of the "definitions" element), and the target namespace of the XML schema in the WSDL ("targetNamespace" attribute of the "xs:schema" element in the "types" element).

- Line numbers 012 and 025

  Use the value of the root element in the SOAP listener definition as the name of the root element of messages to be defined in the XML schema in the WSDL ("name" attribute of the "xs:element" element in "xs:schema" element in the "types" element) and the element defined as a message of the WSDL ("element" attribute of the "part" element in the "message" element).

- Line numbers 015 and 016

  Use the output property name of the complex event processing statement (SELECT statement) as the name of the subelement of the root element to be defined in the XML schema in the WSDL ("name" attribute of the "xs:element" element under the root element definition in the "types" element).

Create as many similar lines as the number of properties to be output. In the example above, this element type ("`type`" attribute of the "`xs:element`" element) is specifying a string ("`xs:string`"). Setting this to suit the property type will, depending on the tool used, generate source code according to the type, so type conversion will no longer be required in the program.

- Line number 047

  In the final WSDL for service publishing, the actual URL of the service will be entered in the "`location`" attribute of the "`soapbind:address`" element under the "`service`" element, and this will also be the value of the connection URL in the listener definition. However, this connection URL is often unsure at development, so there is no problem with leaving it as in the sample above.

2. **Implement the Web service application.**

   Based on the created WSDL, output a template for the Web service application from the development tool being used, and then add the application logic to it.

5.7.3.3 Implementing a User-developed Java Class (Custom Listener)

Implement a user-developed Java class to be called via the custom listener of the CEP engine.

5.7.3.3.1 CustomListener interface

A user-developed Java class must implement the following Java interface:

```
com.fujitsu.cspf.cep.CustomListener
```

The interface is contained in /opt/FJSVcep/cep/lib/CepServerCustom.jar on the CEP Server. Copy CepServerCustom.jar to the Java development environment you are using, set the class path, and develop the Java class.

5.7.3.3.2 Custom log

Logs can be output from a user-developed Java class to the log file (custom log) for the custom listener. The output destination of the custom log is as follows:

```
/var/opt/FJSVcep/cep/cep/logs/EngineLog/cepEngineName/custom.log
```

Output

Use the Apache Log4j class (org.apache.log4j.Logger) for implementation.

Obtain a log output instance using the following method (the argument must be "custom"):

```
Logger myLogger = Logger.getLogger("custom");
```

Information

The Apache Log4j jar file is located in /opt/FJSVcep/log4j/lib/log4j-1.2.16.jar.

Use the fatal, error, warn, and info methods of the Logger class to output logs.

```
myLogger.error("xxxxxx");
```

Note

You cannot use the trace or debug methods of the org.apache.log4j.Logger class.

5.7.3.3.3 Compilation

Compile the source code you created, and generate a class file or jar file.

If using javac of JDK7 or later, specify the "-target 1.6" option when compiling the source code.

5.7.3.4 Implementing an Event Log Analysis Application

Implement an application to analyze event logs. Implement the application using the Hadoop API. Refer to the Interstage Big Data Parallel Processing Server manuals for details.

5.7.3.5 Implementing a Terracotta Application

Implement a Terracotta application. Refer to the Interstage Terracotta BigMemory Max manuals for details.

5.8 Deploying Development Assets

This section explains how to deploy development assets. The deployment tasks are as follows:

- Deploying Definition Information
- Providing Data
- Deploying a Collaboration Application

5.8.1 Deploying Definition Information

Store the developed definition information in the CEP Server, and then use cepdeployrsc to deploy the definition information. Refer to "6.1.3.1 Deploying Definition Information" for information on how to deploy definition information.

Below is an example of deploying definition information.

Example

Example of deploying definition information

This is an example of command execution when the following definition information (definition file) is deployed in a CEP engine (CepEngine):

```
/application
    +-- EVENT01.xml     ... Event type definition    (development asset ID: EVENT01)
    +-- RULE01.xml      ... Rule definition          (development asset ID: RULE01)
    +-- MASTER01.xml    ... Master definition        (development asset ID: MASTER01)
    +-- RDBREF01.xml    ... RDB reference definition (development asset ID: RDBREF01)
    +-- LISTENER01.xml  ... SOAP listener definition (development asset ID: LISTENER01)
```

The example of command execution is as follows:

```
$ cepdeployrsc eventtype -e CepEngine -f /application/EVENT01.xml<ENTER>
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
$ cepdeployrsc rule -e CepEngine -f /application/RULE01.xml<ENTER>
(...)
$ cepdeployrsc master -e CepEngine -f /application/MASTER01.xml<ENTER>
(...)
$ cepdeployrsc rdb_ref -e CepEngine -f /application/RDBREF01.xml<ENTER>
(...)
$ cepdeployrsc listener -e CepEngine -f /application/LISTENER01.xml<ENTER>
(...)
```

5.8.2 Providing Data

Provide the data required for rule operation.

Event data (for testing)

Store event data (for testing) in the event sender system.

If an event sender sample program is to be used as the event sender application, store this data in the CEP Server.

Log storage area

Create a directory to be used as the log storage area, and set the write permission for the engine execution user created during installation. This task is usually performed by the system administrator of the CEP Server.

Information

If the directory to be used as the log storage area does not exist, it will be generated automatically at logging.

Use the engine execution user permission to create the directory.

Example

Directory creation

This is an example of creating a directory when the engine execution user and group are "`isbdcep`".

Log in as a superuser and execute the hadoop command to perform these tasks.

If a user-defined directory name is specified in the directory element in the engine configuration file

Create a directory with the specified name in the root directory of the Hadoop file system.

Change the owner of created directory to give write permissions to the engine execution user.

The log storage area to be specified in the event type definition or logging listener will be created automatically.

Below is an example when "`hadoop`" is specified in the `directory` element.

```
# hadoop fs -mkdir /hadoop <ENTER>
# hadoop fs -chown isbdcep:isbdcep /hadoop <ENTER>
```

If a slash (`/`) only is specified in the directory element in the engine configuration file

No further action is required if the engine execution user can write to the root directory of the Hadoop file system.

If the engine execution user does not have write permissions, create a directory with the same name as the log storage area to be specified in the event type definition or logging listener.

Change the owner of created directory to give write permissions to the engine execution user.

Below is an example when "`/tmp`" is specified as the log storage area to be specified in the event type definition or logging listener.

```
# hadoop fs -mkdir /tmp <ENTER>
# hadoop fs -chown isbdcep:isbdcep /tmp <ENTER>
```

Master data

Store a schema file and data files in the path specified in the master definition.

The stored files must have the file read permission set for the engine execution user created during installation.

Terracotta cache

If a new Terracotta cache is to be provided, use the Terracotta application that was developed separately to store the initial data in the cache.

Note that the Terracotta server must be set beforehand and a cache to be used as the storage destination must also be created beforehand. Refer to the Interstage Terracotta BigMemory Max manuals for information on how to create a cache.

If an existing cache is to be used as it is, no further action required.

Relational database

If preparing a new relational database, use the RDB commands to store initial data in the database.

The relational database where initial data is to be stored must be created in advance. Refer to the manual for the collaboration destination RDB for information on how to create a relational database.

No operation is required if you use an existing relational database as is.

5.8.3 Deploying a Collaboration Application

Deploy the developed collaboration application.

Event sender application

Deploy an event sender application in the event sender system. Perform the deployment according to the application deployment method of the event sender system.

If an event sender sample program is to be used as the event sender application, no deployment is required.

User-developed Web service

Deploy a user-developed Web service in an application server. Perform the deployment according to the method in the manual of the application server to be used.

User-developed Java class

Store the created class file in the following directory:

```
/etc/opt/FJSVcep/config/custom/engineName/classes
```

Store the created jar file in the following directory:

```
/etc/opt/FJSVcep/config/custom/engineName
```

Note

- Grant access permissions so that the engine execution user can read the stored class file and jar file.

- For storing a class file, create a directory corresponding to the class package name in the classes directory. Grant access permissions also for the created directory so that the engine execution user can read it.

- The class file and the jar file stored while the CEP engine is running are not enabled until the CEP engine is restarted.

Example

Example of storing a class file

Create a directory named `com/example` as shown below for storing the `com.example.Example` class.

```
# cd /etc/opt/FJSVcep/config/custom/engineName/classes <ENTER>
# mkdir -p com/example <ENTER>
# chmod 755 com <ENTER>
# chmod 755 com/example <ENTER>
# cp pathOfClassFileToBeStored com/example/ <ENTER>
# chmod 644 com/example/classFileName <ENTER>
```

Event log analysis application

Deploy an event log analysis application in a Hadoop system. Perform the deployment according to the method in the Interstage Big Data Parallel Processing Server manual.

Terracotta application

Deploy a Terracotta server. Perform the deployment according to the method in the Interstage Terracotta BigMemory Max manual.

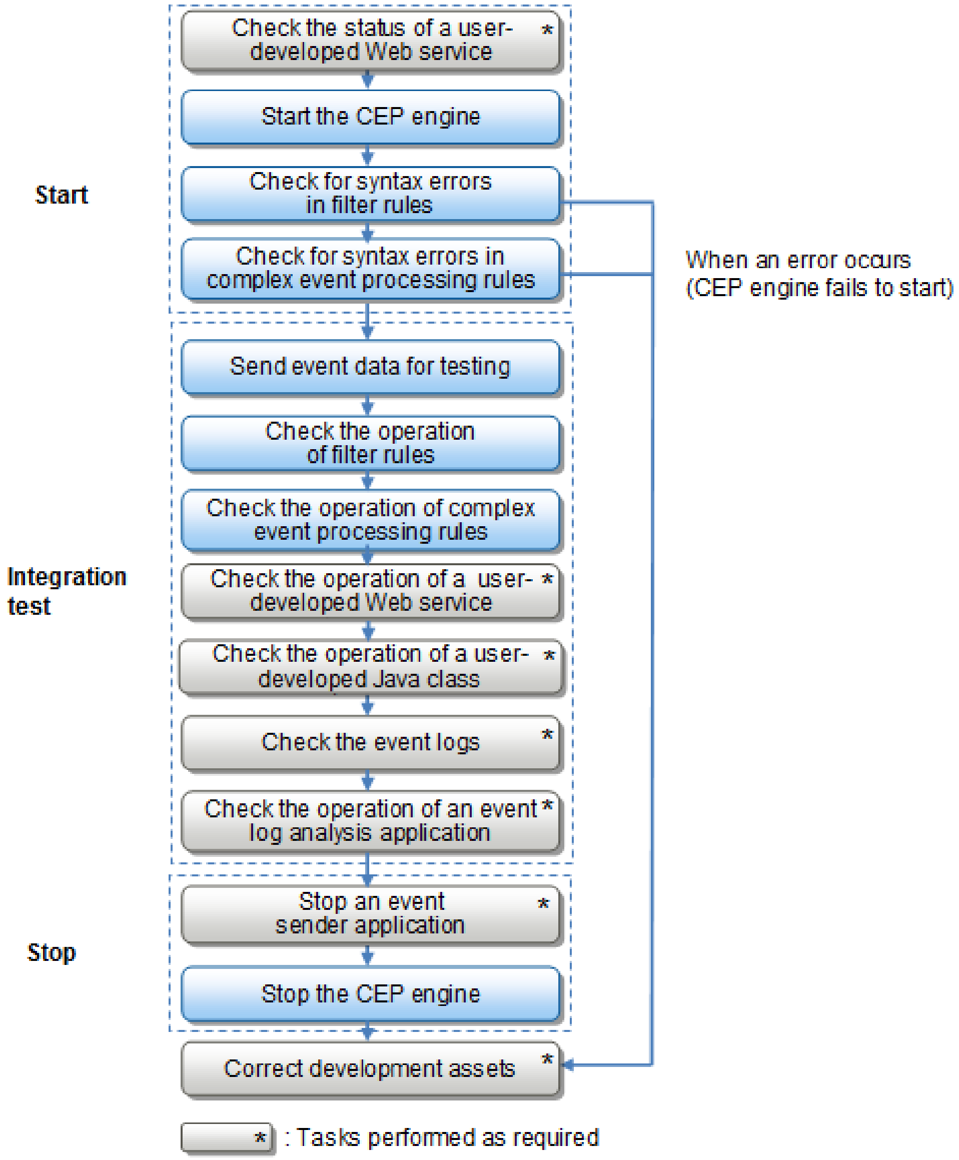
5.9 Integration Test

Send test data to a CEP engine in which development assets have been deployed in order to check operation.

5.9.1 Integration Test Flow

Perform an integration test when the CEP engine is running. Below is the test flow.

Figure 5.6 Integration test flow



Start
Check the status of a user-developed Web service *
Start the CEP engine
Check for syntax errors in filter rules
Check for syntax errors in complex event processing rules
When an error occurs (CEP engine fails to start)
Integration test
Send event data for testing
Check the operation of filter rules
Check the operation of complex event processing rules
Check the operation of a user-developed Web service *
Check the operation of a user-developed Java class *
Check the event logs *
Check the operation of an event log analysis application *
Stop
Stop an event sender application *
Stop the CEP engine
Correct development assets *
* : Tasks performed as required

5.9.2 Checking an Engine Log

The debug information of a CEP engine and the error messages generated at its start or during its operation will be output to an engine log. Rule errors checked when the CEP engine starts will also be output to the engine log.

One CEP engine outputs two engine logs. One is used for output relating to input adapter and high-speed filter processing, and the other is used for output relating to complex event processing and output adapter processing. The engine logs have no predetermined format.

The output destination of each engine log is shown below.

Engine log of the high-speed filter

```
/var/opt/FJSVcep/cep/flt/logs/EngineLog/cepEngineName/engine.log
```

Engine log of complex event processing

```
/var/opt/FJSVcep/cep/cep/logs/EngineLog/cepEngineName/engine.log
```

If "`DebugLogListener`" is used in complex event processing rules, processing results will be output to the engine log of complex event processing.

Note

Engine log splitting

If a "DebugLogListener" log of more than 102,400 characters is to be output at once to the engine log, the log will be split and then output. Refer to "Checking when a DebugLogListener log has been split" below for details.

Checking when a DebugLogListener log has been split

If the output results of hits for complex event processing statement conditions exceed 102,400 characters, the output results will be split every 102,400 characters and then output.

At split output, beginning and end identifiers will be added to the split output log. When this happens, only the end identifier will be output in the first output result of the split log, and only the beginning identifier will be output in the last output result.

Beginning identifier

The following content will be output:

```
YYYY-MM-DD hh:mm:ss,sss [DEBUG]
*****CUT_threadID_nanosecond*****
```

End identifier

The following content will be output. *`threadID`* and *`nanosecond`* will have the same values as those of the beginning identifier.

```
*****CUT_threadID_nanosecond*****
```

Note

Possibility of log output getting out of sequence

There is a possibility that the processing results of other events being executed simultaneously will interrupt the units into which the log was split. Reference the log according to "Example of output when a log has been split" below.

Example

Example of output when a log has been split

```
2011-12-07 14:01:13,720 [DEBUG] 6e1619bd-a048-4c64-9efb-306e9f2b88d6:length=100000
6e1619bd-a048-4c64-9efb-306e9f2b88d6[0]
        residence       :12: String
        value   :ON: String
        gatewayId       :00000001: String
(...)
6e1619bd-a048-4c64-9efb-306e9f2b88d6[123]
        residence:30: String
        val
*****CUT_94_19131119552293*****     ...1(end of the first portion of the split log)

~~~ log of other events ~~~

2011-12-07 14:01:13,720 [DEBUG]     ...2(start of the continuing portion of the split log)
*****CUT_94_19131119552293*****
ue      :ON: String
        gatewayId       :00000001: String
6e1619bd-a048-4c64-9efb-306e9f2b88d6[124]
```

```
        residence       :38: String
        value   :ON: String
        gatewayId       :00000001: String
(...)
6e1619bd-a048-4c64-9efb-306e9f2b88d6[247]
        residence       :30: String
        val
*****CUT_94_19131119552293*****    ...3(end of the continuing portion of the split log)

~~~log of other events ~~~

2011-12-07 14:01:13,720 [DEBUG]    ...4(start of the last portion of the split log)
*****CUT_94_19131119552293*****
ue      :ON: String
        gatewayId       :00000001: String
6e1619bd-a048-4c64-9efb-306e9f2b88d6[248]
        residence       :38: String
        value   :ON: String
        gatewayId       :00000001: String  ...5(end of the last portion of the split log)

~~~log of other events ~~~
```

Explanation of output example:

1. End of the first part of the split output log. Check the identifier (`*****CUT_94_19131119552293*****`).
2. Next beginning of the continuing part of the split output log. This is started using the same identifier (`*****CUT_94_19131119552293*****`).
3. Next end of the continuing part of the split output log. This is split using the same identifier.
4. Similarly, this is split up to the last part using the same identifier.
5. The end of the last part of the split output log has no identifier.

5.9.3 Starting

Start the deployed user-developed Web service and the CEP engine in which definition information has been deployed.

Also check syntax errors along with starting the CEP engine.

The procedure for the starting tasks is as follows:

1. Checking the Status of a User-developed Web Service
2. Starting the CEP Engine
3. Checking for Syntax Errors in Filter Rules
4. Checking for Syntax Errors in Complex Event Processing Rules

5.9.3.1 Checking the Status of a User-developed Web Service

If the application server in which a user-developed Web service has been deployed is not running, start the server. After starting the server, check that the deployed user-developed Web service has the status of receiving requests from outside.

Refer to the application server manuals for information on how to start the application server and how to check the status of the Web service.

5.9.3.2 Starting the CEP Engine

After deployment of development assets, use cepstarteng to start the CEP engine.

Refer to "5.8 Deploying Development Assets" for information on deploying development assets.

Refer to "8.9 cepstarteng" for information on the cepstarteng command.

Note

If implementing RDB collaboration or Terracotta collaboration, ensure that all collaboration destination servers are started before the CEP engine is started.

Example

Example of executing cepstarteng

```
$ cepstarteng -e CepEngine<ENTER>
Command cepstarteng executed successfully.
```

5.9.3.3 Checking for Syntax Errors in Filter Rules

If the filter rules in the rule definition contain syntax errors, the CEP engine start will fail. If this happens, the cause of the error will be notified to the engine log of the high-speed filter. Correct the syntax error based on the notified error content.

Redeploy the corrected rule definition in the CEP engine. Refer to "6.1.3.3 Updating Deployed Definition Information" for information on redeploying. After redeploying the rules, repeat the operations from "5.9.3.2 Starting the CEP Engine".

5.9.3.4 Checking for Syntax Errors in Complex Event Processing Rules

If the complex event processing statements in the rule definition contain syntax errors, the CEP engine start will fail with a "cep20201e" error.

If this happens, the cause of the error will be notified to the engine log of complex event processing using the error message in the "cep20201e" ERRORINFO parameter.

Below is an example of an error message in the ERRORINFO parameter. Correct the syntax error based on this content.

```
Incorrect syntax near xxxxxxxx at line X column Y
```

xxxxxxxx: Keyword near the abnormality

X: Number of lines from the beginning of the complex event processing statement in which the error occurred

Y: Number of characters from the beginning of the line of the complex event processing statement in which the error occurred

Redeploy the corrected rule definition in the CEP engine. Refer to "6.1.3.3 Updating Deployed Definition Information" for information on redeploying. After redeploying the rules, repeat the operations from "5.9.3.2 Starting the CEP Engine".

Note

Depending on the content of the error message, the location of the error (line and column information) may not be displayed.

Example

Log output example of a syntax error in a complex event processing statement

Target complex event processing statement

```
@Name('EPL3')
@DebugLogListener
select * from  FilteredCouponEvent (storeID='STR0001') wherer cast(targetAge,int)>20;
```

Log output (Note that in the example below, newlines have been added for readability only (lines 1 to 6). The actual output does not have newlines.)

```
2012-07-15 13:51:18,843 [ERROR] CSPF_CEP: ERROR: cep20201e: EPL module file access
failure. EngineId=CepEngine, FILE=/etc/opt/FJSVcep/resources/CepEngine/rules/SampleRule.
epl, ERRORINFO=com.espertech.esper.client.deploy.DeploymentActionException: Compilation 
failed in module url '/etc/opt/FJSVcep/resources/CepEngine/rules/SampleRule.epl' in
expression '@Name('EPL3')@DebugLogListenerselect * from Filt...(115 chars)' : Incorrect 
syntax near 'cast' (a reserved keyword) at line 3 column 61, please check the from
clause [@Name('EPL3')
@DebugLogListener
select * from FilteredCouponEvent (storeID='STR0001') wherer cast(targetAge,int)>20]
```

Explanation of log output:

- `ERRORINFO=...`

  Error information will be output following this `ERRORINFO` parameter.

- `Compilation failed...`

  This is a broad error classification. "`Compilation failed`" or "`Deployment failed`" will be output.

- `SampleRule`

  This is the development asset ID of the rule definition in which the error occurred.

- `Incorrect syntax...`

  This is the error message. It indicates that there is a syntax error near "`cast`" in line 3, column 61 from the beginning of complex event processing statement in which the error occurred.

- `@Name('EPL3')...`

  This is the complex event processing statement in which the error occurred. In this example, the "`wherer`" before "`cast`" should actually be "`where`", and the extra "`r`" has caused a syntax error.

5.9.4 Integration Test

The flow of the integration test tasks is as follows:

1. Sending Event Data for Testing
2. Checking the Operation of Filter Rules
3. Checking the Operation of Complex Event Processing Rules
4. Checking the Operation of a User-developed Web Service
5. Checking the Operation of a User-developed Java Class
6. Checking the Event Log
7. Checking the Operation of an Event Log Analysis Application

5.9.4.1 Sending Event Data for Testing

Send event data for testing to the CEP engine.

Execute the deployed event sender application or the event sender sample program supplied with the samples.

After sending the events, check the engine log to see if event sending is operating as expected.

Refer to "5.11.6 Event Sender Sample Program" for information on how to use an event sender sample program.

Example

Sending event data for testing

This is an example of sending the "/application/test.csv" data for testing, which is in CSV format, to a CEP engine (CepEngine) as event data with the event type "EVENT01".

```
$ /opt/FJSVcep/sample/sample1/bin/sendevent.sh EVENT01 /application/test.csv<ENTER>
```

5.9.4.2 Checking the Operation of Filter Rules

Check whether any errors have been notified to the engine log of the high-speed filter, and correct the error based on the notified error content.

In addition to errors, also check whether any content on input events (those with logging enabled in the event type definition) was logged in the engine log, if logging with the "file" logging type is being used.

5.9.4.3 Checking the Operation of Complex Event Processing Rules

Check whether any errors have been output to the engine log of complex event processing, and correct the error based on the notified error content.

In addition to errors, also check the operation of the complex event processing rules by checking the engine log for any content output using the @DebugLogListener annotation described in a complex event processing statement.

If the conditions in the complex event processing rule with the @DebugLogListener annotation have a hit, the property value of the output event will be output to the log.

Also check whether any content output using the @LoggingListener annotation was logged in the engine log, if logging with the "file" logging type is being used.

Note

The following considerations relate to the operation results of complex event processing rules:

- Once checking the operation results of the rules is complete, before deploying the rule definition in the production environment, edit it to delete the @DebugLogListener annotation from the rule definition.
- To correct and replace a rule definition during testing, first stop the CEP engine, then redeploy the rule definition, and then restart the CEP engine.
- When the CEP engine is stopped, any data that was in the windows generated by rules before it was stopped will disappear.

Example

Log output example of "DebugLogListener"

Below is an example of engine log output using the @DebugLogListener annotation, and will be the execution result of the following complex event processing statement:

Complex event processing statement to be used

```
@Name('EPL3')
@DebugLogListener
@LoggingListener(table='/EPL3', properties='areaID, couponID')
@SoapListener('LISTEN01')
select areaID, targetAge, storeID, couponID
  from FilteredCouponEvent (storeID='STR0001') where cast(targetAge,int)>20;
```

Log output example of "DebugLogListener"

```
2012-07-15 14:21:11,854 [DEBUG] EPL3:length=1    ... 1.
EPL3[0]                                          ... 2.
        areaID  :2222: String                    ... 3.
        targetAge       :30: String
        storeID :STR0001: String
        couponID        :CPN0001: String
```

Explanation of output example:

1. This displays the number of records that are a hit from the SELECT statement condition. The value specified using the `@Name` annotation appears in the underlined part.
2. This displays the record index. The value specified using the `@Name` annotation appears in the underlined part.
3. This line onwards displays the properties, property values, and property types of the output event.

Information

Example of an engine log output by other output adapters

An engine log output by other output adapters can also be used as debug information because it shows whether listener calling succeeded.

Example of engine log output of the SOAP listener

If the conditions in the complex event processing statement associated with the `@SoapListener` annotation have a hit, the successful listener calling will be output to the engine log. (Note that in the example below, newlines have been added for readability only. The actual output does not have newlines.)

```
2012-07-15 14:44:42,556 [DEBUG] id=LISTEN01 destination=http://xxx.xxx.xx.xxx/WebServWAR
/MyApp1Service methodName=root<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.
org/soap/envelope/"><SOAP-ENV:Header/><SOAP-ENV:Body><ns:root xmlns:ns="http://webservic
e/"><ns:areaID>2222</ns:areaID><ns:targetAge>30</ns:targetAge><ns:storeID>STR0001</ns:st
oreID><ns:couponID>CPN0001</ns:couponID></ns:root></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Explanation of output example:

- `LISTEN01`

  This is the development asset ID of the SOAP listener definition that was used.

- `http://xxx.xxx.xx.xxx/WebServWAR/MyApp1Service`

  This is the connection URL. This will be the value specified in the "`url`" tag in the SOAP listener definition.

- `root`

  This is the root element name. This will be the value specified in the "`method`" tag in the SOAP listener definition.

- From "`root`" onwards

  This is the SOAP message that was sent.

Example of engine log output of the logging listener

If the conditions in the complex event processing statement associated with the `@LoggingListener` annotation have a hit, the successful logging will be output to the engine log.

```
2012-07-26 00:05:18,692 [DEBUG] Write Log message"2222","CPN0001"
```

If the engine log was used as the logging destination (by specifying "`file`" in "`type`" in the engine configuration file), the processing results of the complex event processing rules will also be output.

```
2012-07-26 00:05:18,692 [DEBUG] TableName:/EPL3; eventdata:"2222","CPN0001"
2012-07-26 00:05:18,692 [DEBUG] Write Log message"2222","CPN0001"
```

5.9.4.4 Checking the Operation of a User-developed Web Service

Perform this only if a user-developed Web service has been designed and developed.

Check the operation of a user-developed Web service based on information such as the logs it outputs.

5.9.4.5 Checking the Operation of a User-developed Java Class

Perform this step only if you designed and developed a user-developed Java class. Check the operation based on the custom log output by the user-developed Java class. The output destination of the custom log is as follows:

```
/var/opt/FJSVcep/cep/cep/logs/EngineLog/cepEngineName/custom.log
```

5.9.4.6 Checking the Event Log

If a setting to perform logging using Hadoop collaboration has been specified, check that an event log has been logged in the Hadoop system.

5.9.4.7 Checking the Operation of an Event Log Analysis Application

Perform this only if an event log analysis application has been designed and developed.

Use a recorded event log to check the operation of an event log analysis application.

5.9.5 Stopping

Stop the deployed event sender application and the CEP engine in which definition information has been deployed.

5.9.5.1 Stopping an Event Sender Application

If an event sender application is still sending events, stop the event sender application.

5.9.5.2 Stopping the CEP Engine

Stop the CEP engine. Use cepstopeng to stop the CEP engine.

Refer to "8.11 cepstopeng" for details.

Example

Example of command execution

This is an example of command execution when stopping a CEP engine (CepEngine).

```
$ cepstopeng -e CepEngine<ENTER>
Command cepstopeng executed successfully.
```

5.9.6 Correcting Development Assets

If the checked operation results have a problem, correct the development asset.

After correcting it, first undeploy the development asset, then redeploy it, and then perform an integration test again.

Point

Updating definition information by overwriting

If only part of the definition information is to be corrected, using the `-o` option of the cepdeployrsc command allows the definition information to be updated by overwriting (and redeployed).

5.10 Undeploying Development Assets

Undeploy any development assets that are no longer required.

5.10.1 Undeploying Definition Informations

Execute cepundeployrsc to undeploy definition information that is no longer required.

Refer to "8.13 cepundeployrsc" for details.

Below is an example of undeploying.

Note

Note on executing cepundeployrsc

Definition information cannot be undeployed while the CEP engine is running.

Example

Example of undeploying definition information

This is an example of command execution when the following definition information that has been deployed in a CEP engine (CepEngine) is to be undeployed:

Type of definition information	Development asset ID
Event type definition	EVENT01
Rule definition	RULE01
Master definition	MASTER01
RDB reference definition	RDBREF01
SOAP listener definition	LISTENER01

The command execution example is as follows:

```
$ cepundeployrsc eventtype -e CepEngine -n EVENT01<ENTER>
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
$ cepundeployrsc rule -e CepEngine -n RULE01<ENTER>
(...)
$ cepundeployrsc master -e CepEngine -n MASTER01<ENTER>
(...)
$ cepundeployrsc rdb_ref -e CepEngine -n RDBREF01<ENTER>
(...)
$ cepundeployrsc listener -e CepEngine -n LISTENER01<ENTER>
(...)
```

5.10.2 Undeploying a Collaboration Application

Undeploy any collaboration application that is no longer required.

Event sender application

Undeploy an event sender application according to the application undeployment method of the event sender system.

If an event sender sample program is being used as the event sender application, no undeployment is required.

User-developed Java class

Manually delete the class file and the jar file of the user-developed Java class from the deployment destination directory.

User-developed Web service

Undeploy a user-developed Web service from the application server. Perform the undeployment according to the method in the application server manual.

Event log analysis application

Undeploy an event log analysis application from the Hadoop system. Perform the undeployment according to the method in the Interstage Big Data Parallel Processing Server manual.

Terracotta application

Delete the application from the Terracotta server. Perform the deletion according to the method in the Interstage Terracotta BigMemory Max manual.

5.10.3 Deleting Data

Delete any data that is no longer required.

Event data (for testing)

Delete event data for testing from the event sender system.

If an event sender sample program is being used as the event sender application, delete the event data for testing that was stored on the CEP Server.

Logging storage destination

Delete the directory that was used as the logging storage destination.

If the data in the directory is required, store the data in another location and then delete the directory.

Master data

Delete the master data that was stored in the path specified in the master definition.

Terracotta cache

Refer to the Terracotta manuals for information on how to delete a Terracotta cache.

Relational database

Refer to the manual for the collaboration destination RDB for information on how to delete a relational database.

5.11 Sample Application

This section explains the sample application supplied with BDCEP.

5.11.1 Overview of the Sample Application

This sample is an application for detecting members walking in the vicinity of a registered store and issuing them with coupons that have a time limit.

The application for this service consists of the definition information, which is the part to be processed by the CEP engine, and the master data, which is to be used by the rules. Sample event data and an event sender sample program are also supplied for checking the operation of the application.

The following is an overview of the coupon delivery service contained in the sample application.

Entities

The main entities of the service are as follows:

- The coupon delivery service
- Registered service members
- Registered service stores

Overview of the coupon delivery service

Process flow of the assumed service is as follows:

- Location information of the registered members is periodically sent to the service from their smartphones.
- The registered stores send a request to issue coupons usable during a specific time to the service.
- Once the service receives this request, it will issue the coupons to the members walking around the stores.

In the sample, the event sender sample program is used to send sample event data instead of actual registered users and stores sending events, and the performance is checked using CEP engine logs.

Data type

Data handled by the service is as follows:

- Event data
  - Location information (location information event)

    Contains information such as member IDs, IDs of the area where the member is currently at, and the member status (walking, moving on train etc.).
  - Coupon issue request (coupon issue event)

    Contains information such as store IDs, coupon IDs, and delivery target age groups.
- Master data
  - Member information (member master data)

    Contains information such as member IDs and age groups.
  - Store information (store master data)

    Contains information such as store IDs and IDs of the area where the store is located.

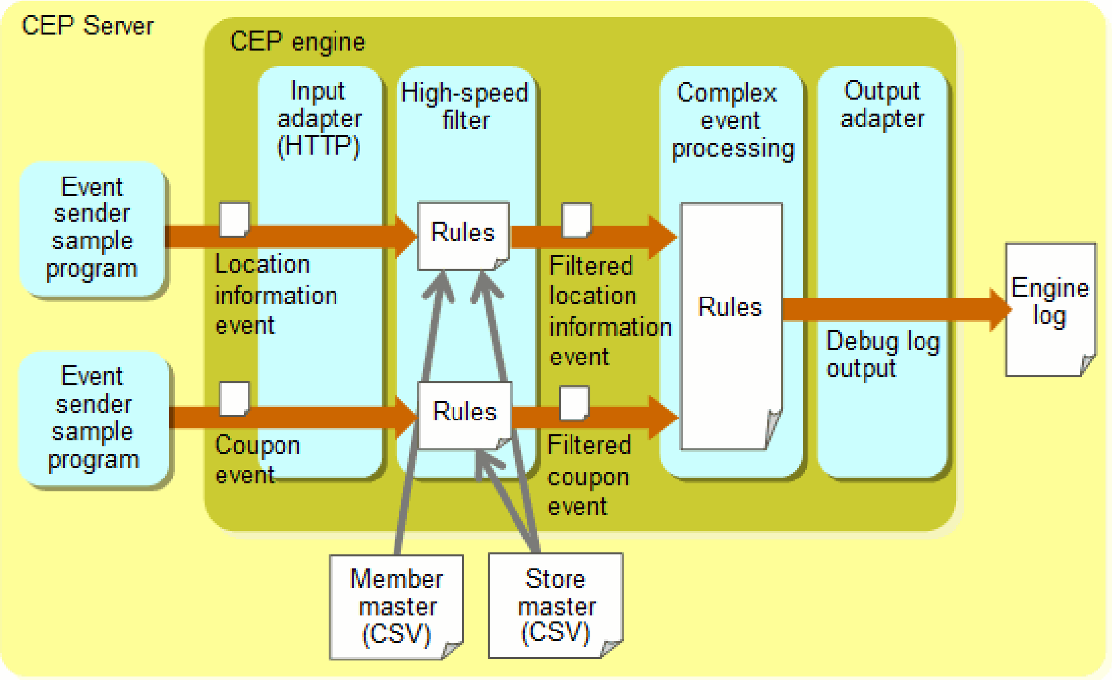
5.11.2 Structure of the Sample

The table below shows the structure of the development assets (and what is provided) in this sample.

Development asset type	Development asset	What is provided
Definition file	Event type definition	Location information event definition Coupon event definition
	Rule definition	Sample rule definition
	Event type definition (filtered events)	Filtered location information event definition Filtered coupon event definition
	Master definition	Member master definition Store master definition
Data	Master data (CSV file)	Member master data Store master data
	Event data	Sample data for location information events Sample data for coupon events
Collaboration application	Event sender application	Event sender sample program (Binary and source code)

Below is an example of how the sample operates.

Figure 5.7 Example of how the sample operates



5.11.3 Events

5.11.3.1 Location Information Events

The table below shows the content of the event type definition for location information events.

Identifier (development asset ID)	Format	Column	
		Name	Description
LocationEvent	CSV	memberID	Member ID. The format of the sample data is "MEM" followed by four digits.
		areaID	Area code of the area where the member is currently located. The format of the sample data is four digits.
		status	Current status of the member (walking: "1", moving on train: "2").

5.11.3.2 Coupon Events

The table below shows the content of the event type definition for coupon events.

Identifier (development asset ID)	Format	Column	
		Name	Description
CouponEvent	CSV	storeID	Store ID. The format of the sample data is "STR" followed by four digits.
		couponID	Coupon ID. The format of the sample data is "CPN" followed by four digits.
		targetAge	Delivery target age. (10's: "10", 20's: "20", 30's: "30", 40's: "40", 50's: "50", 60 and over: "60")

5.11.3.3 Filtered Location Information Events

Filter rules for location information events perform the following processes:

- Extracts only events where the status is "walking".
- Extracts only events that occurred in an area where registered stores exist, based on the "areaID" and store master information.
- Joins the member age information from the member information master for complex event processing with the location events.
- Does not output the status information as it is no longer used after filter processing.

The table below shows the content of the event type definition for filtered location information events.

Identifier (development asset ID)	Format	Column	
		Name	Description
FilteredLocationEvent	CSV	memberID	Member ID.
		areaID	Area code of the area where the member is currently located.
		age	Member age. (10's: "10", 20's: "20", 30's: "30", 40's: "40", 50's: "50", 60 and over: "60")

5.11.3.4 Filtered Coupon Events

Filter rules for coupon events perform the following process:

- Joins the store area code from the store information master for complex event processing with the coupon information events.

The table below shows the content of the event type definition for filtered coupon events.

Identifier (development asset ID)	Format	Column	
		Name	Description
FilteredCouponEvent	CSV	storeID	Store ID.
		couponID	Coupon ID.
		targetAge	Delivery target age. (10's: "10", 20's: "20", 30's: "30", 40's: "40", 50's: "50", 60 and over: "60")
		areaID	Area code of the area where the store is located.

5.11.4 Master Information

This section explains the master data to be used by this sample.

5.11.4.1 Member Information Master

This is the master data of the registered member, which stores the member ID and member age data.

The table below shows the content of the member information master definition.

Identifier (development asset ID)	Column	
	Name	Description
MemberInfo	ID	Member ID.
	age	Member age.

Identifier (development asset ID)	Column	
	Name	Description
		(10's: "10", 20's: "20", 30's: "30", 40's: "40", 50's: "50", 60 and over: "60")

5.11.4.2 Store Information Master

This is the master data of the registered store, which stores the store ID and area code of the area where the store is located.

The table below shows the content of the store information master definition.

Identifier (development asset ID)	Column	
	Name	Description
StoreInfo	ID	Store ID.
	areaID	Area code of the area where the store is located.

5.11.5 Rule Definition

The rule definition processes two types of events. It consists of filter rules and complex event processing rules.

5.11.5.1 Filter Rules (IF-THEN Format)

These filter rules are for two types of events.

```
 1   on LocationEvent {
 2     if ($status == "1" AND lookup("StoreInfo", $areaID == $areaID) = true()) then
 3       join("MemberInfo", $memberID==$ID)
 4         output($memberID, $areaID, "MemberInfo".$age) as FilteredLocationEvent;
 5   }
 6
 7   on CouponEvent {
 8     join("StoreInfo", $storeID == $ID)
 9       output($storeID, $couponID, $targetAge, "StoreInfo".$areaID) as FilteredCouponEvent;
10   }
```

Below is an explanation of each line.

- Line 1

  This is a filter rule for a location information event (`LocationEvent`).

- Line 2

  This search expression extracts only the events where a registered store can be found in the area (one with an "`areaID`" equal to the "`areaID`" in the "`storeInfo`" store master) while walking ("`status`" is "`1`"). The left side of "`$areaID == $areaID`" is the "`areaID`" of the input events, and the right side is the "`areaID`" of the master data.

- Line 3

  To assign member master information to the input events, this uses the data of the member master "`MemberInfo`" to join the events that have equal member IDs.

- Line 4

  This outputs the member ID and area ID that are in the input events, as well as the age information of the member master, as a "`FilteredLocationEvent`".

- Line 7

  This is a filter rule for a coupon event (`CouponEvent`).

- Line 8

  This uses the store master "StoreInfo" data to join the events that have equal store IDs.

- Line 9

  This outputs the store ID, coupon ID, and target age that are in the input events, as well as the area ID of the store master, as a "FilteredCouponEvent".

5.11.5.2 Complex Event Processing Rules (SQL Format)

These complex event processing rules are for the two types of events processed by the filter rules.

```
 1   create window outEventWin.std:firstunique(memberID,couponID).win:time(3 min)
 2     as (memberID string, storeID string, couponID string);
 3
 4   @Name('EPL1')
 5   insert into outEventWin
 6     select loc.memberID as memberID,cpn.storeID as storeID,cpn.couponID as couponID
 7       from FilteredLocationEvent as loc unidirectional
 8            inner join FilteredCouponEvent.win:time(3 min) as cpn
 9         on loc.areaID=cpn.areaID and loc.age=cpn.targetAge;
10
11   @Name('EPL2')
12   @DebugLogListener
13   select * from outEventWin;
14
15   @Name('filterOut1')
16   @DebugLogListener
17   select * from FilteredLocationEvent;
18
19   @Name('filterOut2')
20   @DebugLogListener
21   select * from FilteredCouponEvent;
```

Below is an explanation of each line.

- Line 1

  This is a complex event processing statement that creates a named window to retain the output events for a set time period and to ensure that the same coupon is not sent to the same person twice within that period. This retains the first output events for member ID and coupon ID pairs, one at a time. After the set period (3 minutes in the sample) has elapsed, they are deleted from the window.

- Line 2

  This defines the format of events held in a named window.

- Line 4

  This defines the complex event processing statement to perform the main process. Its name is "EPL1".

- Line 5

  This inserts the output of the main process in the named window.

- Line 6

  This outputs the member IDs of location information events and the store IDs and coupon IDs of coupon events, as the results of the main process.

- Line 7

  This is for the "FilteredLocationEvent", which is one source of input for the main process. When "unidirectional" is specified, the reception of this event will be the trigger for operating this process ("FilteredCouponEvent" will not be the trigger).

- Line 8

  This is for the "FilteredCouponEvent", which is the other source of input for the main process. This event is retained in memory for a set period only (3 minutes in the sample) and is joined to "FilteredLocationEvent".

- Line 9

  The join condition for "FilteredLocationEvent" and "FilteredCouponEvent" is that the "`areaID`s" are equal and that "`age`" and "`targetAge`" are equal.

- Line 11

  This is a complex event processing statement for outputting the processing results of "`EPL1`" (line 5 onwards). Its name is "`EPL2`".

- Line 12

  By assigning a debug log listener, this outputs the output events to the engine log.

- Line 13

  This outputs only the new events registered in the named window. Even if there are more than one of the same event entered in the named window, specifying "`std:firstunique`" (in "`create window`") will ensure that only the first event will actually be stored.

- Lines 15 to 17

  This is a complex event processing statement for performing debug log output of a filtered location information event (`FilteredLocationEvent`). This is for checking the filtering process and can safely be deleted.

- Lines 19 to 21

  This is a complex event processing statement for performing debug log output of a filtered coupon event (`FilteredCouponEvent`). This is for checking the filtering process and can safely be deleted.

5.11.6 Event Sender Sample Program

The event sender sample program is an event sender application that sends event data for testing in CSV format to an HTTP adapter.

Execution format

The execution format of the program is as follows:

```
/opt/FJSVcep/sample/sample1/bin/sendevent.sh eventType dataFilePath [ sendDestinationURL ]
```

Arguments

eventType

Specify which event type is to be used to send the data.

Specify the development asset ID of the target event type.

dataFilePath

Specify the path of the CSV format file that contains the event data.

sendDestinationURL

Specify the endpoint address of the HTTP adapter.

If this is omitted, the default is "`http://localhost/CepEngineFrontServerService/HttpReceiver`".

Information

Endpoint address of the CEP engine

The send destination URL of the CEP engine is as follows:

```
http://CEPengineAddress/cepEngineNameFrontServerService/HttpReceiver
```

The send destination URL of the CEP engine (CepEngine) of the CEP Server ("cephost.example.com") is as follows:

```
http://cephost.example.com/CepEngineFrontServerService/HttpReceiver
```

Source code

Below is the source code of the event sender sample program.

/opt/FJSVcep/sample/sample1/src/sample/HttpClient.java

```java
package sample;

import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class HttpClient {

static String DEFAULT_URL = "http://localhost/CepEngineFrontServerService/HttpReceiver";

public static void main(String[] args) {
   if (args.length != 2 && args.length != 3) {
      System.err.println("Requires two or three arguments: event-type, file name, [URL]");
      System.exit(1);
   }
   String eventType = args[0];
   String fileName = args[1];
   String url = DEFAULT_URL;
   if (args.length == 3) {
      url = args[2];
   }

   try {
      BufferedReader br0 =
         new BufferedReader(new InputStreamReader(new FileInputStream(fileName)));

      String str;
      while ((str = br0.readLine()) != null) {
         URL TestURL = new URL(url);
         HttpURLConnection con = (HttpURLConnection) TestURL.openConnection();

         con.setRequestMethod("POST");
         con.setRequestProperty("TYPE", "CSV");
         con.setRequestProperty("EVENT-TYPE-ID", eventType);
         con.setRequestProperty("Content-Type", "text/plain; charset=utf-8");
         con.setDoOutput(true);

         BufferedOutputStream bos = new BufferedOutputStream(con.getOutputStream());
         bos.write(str.getBytes("utf-8"));
         bos.flush();
         bos.close();

         BufferedReader br1 = new BufferedReader(new InputStreamReader(con.getInputStream()));

         String line;
         while ((line = br1.readLine()) != null) {
            System.out.println(line);
         }
         br1.close();
         con.disconnect();
      }
      br0.close();
   } catch (Exception e) {
      e.printStackTrace();
   }
}
}
```

5.11.7 Directory Structure

This sample is stored in "`/opt/FJSVcep/sample`" using the following structure:

```
sample1
+-- client.jar                 Jar file to be used by the event sender sample program
+-- bin                        Directory for shell scripts
|   +-- deployall.sh                Shell script to deploy all sample development assets
|   +-- undeployall.sh              Shell script to undeploy all sample development assets
|   +-- sendevent.sh                Event sender sample program
+-- src                        Directory for source code
|   +-- sample
|       +-- HttpClient.java    Source code of jar to be used by the event sender sample program
+-- master                     Directory for master data
|   +-- MemberData.csv              Sample data of member master
|   +-- MemberDataSchema.csv        Schema file of member master
|   +-- StoreData.csv               Sample data of store master
|   +-- StoreDataSchema.csv         Schema file of store master
+-- event                      Directory for event data
|   +-- CouponEvent.csv             Sample data of coupon event
|   +-- LocationEvent.csv           Sample data of location information event
+-- resources                  Directory for definition files
    +-- CouponEvent.xml             Event type definition of coupon event
    +-- FilteredCouponEvent.xml     Event type definition of filtered coupon event
    +-- LocationEvent.xml           Event type definition of location information event
    +-- FilteredLocationEvent.xml   Event type definition of filtered location information event
    +-- MemberInfo.xml              Master definition of member master
    +-- StoreInfo.xml               Master definition of store master
    +-- SampleRule.xml              Rule definition
```

5.11.8 Execution

This section explains how to execute the sample application.

Use general user permissions to execute the commands in the tasks below.

Perform the tasks in the following sequence:

1. Deploying Development Assets
2. Starting the CEP Engine
3. Sending Events and Checking the Results
4. Stopping the CEP Engine
5. Undeploying Development Assets

5.11.8.1 Deploying Development Assets

Deploy the development assets when only one CEP engine has been deployed (for example, immediately after the initial setup of BDCEP has completed).

Execute "`/opt/FJSVcep/sample/sample1/bin/deployall.sh`".

During this process, a number of queries will be made, so press "Enter" for all of them.

```
$ cd /opt/FJSVcep/sample/sample1/bin <ENTER>
$ ./deployall.sh <ENTER>
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
Are you sure you want to deploy the rule definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
Are you sure you want to deploy the master data definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
Are you sure you want to deploy the master data definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
```

In this shell script, the following commands are executed:

- Deploying event type definitions

```
cepdeployrsc eventtype -f /opt/FJSVcep/sample/sample1/resources/LocationEvent.xml
cepdeployrsc eventtype -f /opt/FJSVcep/sample/sample1/resources/CouponEvent.xml
cepdeployrsc eventtype -f /opt/FJSVcep/sample/sample1/resources/FilteredLocationEvent.xml
cepdeployrsc eventtype -f /opt/FJSVcep/sample/sample1/resources/FilteredCouponEvent.xml
```

- Deploying master definitions

```
cepdeployrsc master -f /opt/FJSVcep/sample/sample1/resources/MemberInfo.xml
cepdeployrsc master -f /opt/FJSVcep/sample/sample1/resources/StoreInfo.xml
```

- Deploying the rule definition

```
cepdeployrsc rule -f /opt/FJSVcep/sample/sample1/resources/SampleRule.xml
```

5.11.8.2 Starting the CEP Engine

Execute the following command to start the CEP engine:

```
$ cepstarteng<ENTER>
```

5.11.8.3 Sending Events and Checking the Results

Use the event sender sample program (`/opt/FJSVcep/sample/sample1/bin/sendevent.sh`) to send the sample data and use the engine log to check the rule operation.

1. **Sending location information events and checking the results**

   ```
   $ ./sendevent.sh LocationEvent /opt/FJSVcep/sample/sample1/event/LocationEvent.csv<ENTER>
   ```

   Check that "`filterOut1`" events have been output to the engine log.

   Output example

   ```
   2012-07-09 11:11:57,409 [DEBUG] filterOut1:length=1
   filterOut1[0]
           areaID  :2222: String
           age     :30: String
           memberID        :MEM0008: String

   2012-07-09 11:11:57,413 [DEBUG] filterOut1:length=1
   filterOut1[0]
           areaID  :2222: String
           age     :20: String
   ```

```
        memberID        :MEM0003: String

```

2. **Sending coupon events and checking the results**

```
$ ./sendevent.sh CouponEvent /opt/FJSVcep/sample/sample1/event/CouponEvent.csv<ENTER>
```

Check that "`filterOut2`" events have been output to the engine log.

Output example

```
2012-07-09 11:15:38,054 [DEBUG] filterOut2:length=1
filterOut2[0]
        areaID  :2222: String
        targetAge       :30: String
        storeID :STR0001: String
        couponID        :CPN0001: String

```

3. **Sending location information events and checking the results**

Send location information events within 3 minutes of sending coupon events.

```
$ ./sendevent.sh LocationEvent /opt/FJSVcep/sample/sample1/event/LocationEvent.csv<ENTER>
```

Check that events indicating a coupon issue ("`EPL2`" event) have been output to the engine log.

Output example

```
2012-07-09 11:17:25,314 [DEBUG] filterOut1:length=1
filterOut1[0]
        areaID  :2222: String
        age     :20: String
        memberID        :MEM0003: String

2012-07-09 11:17:25,321 [DEBUG] filterOut1:length=1
filterOut1[0]
        areaID  :2222: String
        age     :30: String
        memberID        :MEM0008: String

2012-07-09 11:17:25,323 [DEBUG] EPL2:length=1
EPL2[0]
        storeID :STR0001: String
        memberID        :MEM0008: String
        couponID        :CPN0001: String

```

4. **Sending location information events and checking the results**

Send location information events again within 3 minutes of sending coupon events.

```
$ ./sendevent.sh LocationEvent /opt/FJSVcep/sample/sample1/event/LocationEvent.csv<ENTER>
```

Check that events indicating coupon issue ("`EPL2`" event) have not been output to the engine log.

Output example

```
2012-07-09 11:18:00,691 [DEBUG] filterOut1:length=1
filterOut1[0]
        areaID  :2222: String
        age     :20: String
        memberID        :MEM0003: String

2012-07-09 11:18:00,691 [DEBUG] filterOut1:length=1
filterOut1[0]
```

```
        areaID  :2222: String
        age     :30: String
        memberID        :MEM0008: String
```

5.11.8.4 Stopping the CEP Engine

Execute the following command to stop the CEP engine:

```
$ cepstopeng<ENTER>
```

5.11.8.5 Undeploying Development Assets

Execute "`/opt/FJSVcep/sample/sample1/bin/undeployall.sh`" to undeploy the development assets in batch.

During this process, a number of queries will be made, so press "Enter" for all of them.

```
$ ./undeployall.sh<ENTER>
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
Are you sure you want to undeploy the rule definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
Are you sure you want to undeploy the master data definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
Are you sure you want to undeploy the master data definition?(default: y) [y,n,q]:<ENTER>
Command cepundeployrsc executed successfully.
```

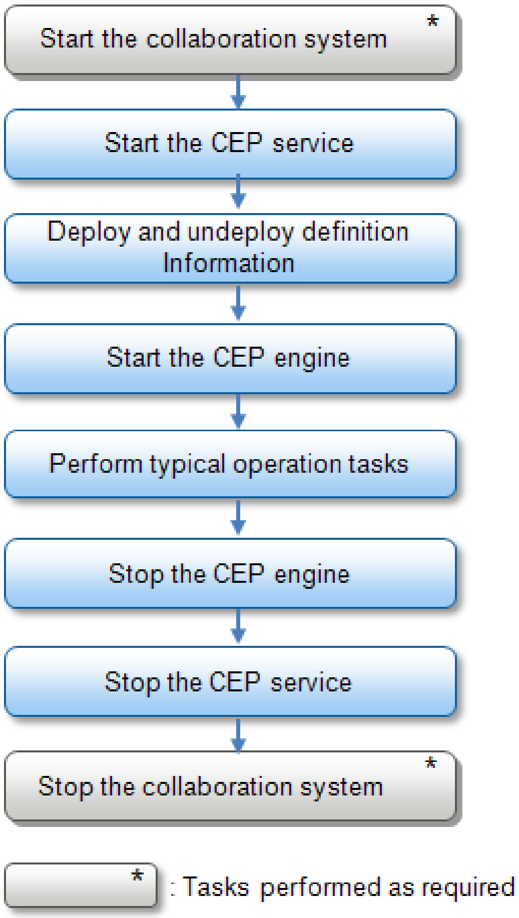
Chapter 6 Operation and Maintenance

This chapter explains how to operate the CEP Server.

If Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP") is to be operated using a reliable configuration, refer to "7.2.4 Operating a Cluster Service".

6.1 Operating the CEP Server

This section explains how to operate the CEP Server.


Start the collaboration system *
Start the CEP service
Deploy and undeploy definition Information
Start the CEP engine
Perform typical operation tasks
Stop the CEP engine
Stop the CEP service
Stop the collaboration system *
* : Tasks performed as required

6.1.1 Starting the Collaboration System

If Terracotta collaboration, RDB collaboration, or Hadoop collaboration is to be performed, first start the relevant collaborating system.

Refer to the manual for the relevant product for information on how to start the system.

6.1.2 Starting the CEP Service

Execute cepstartserv as a superuser to start the CEP service.

Information

- The CEP service consists of multiple services listed below. Execute cepstartserv to start all the services simultaneously.
    - Interstage Java EE DAS service
    - Interstage Java EE Node Agent service

- Interstage Management Console

- Interstage HTTP Server

- PostgreSQL

- Apache Tomcat

Note

Automatic start of the CEP service and CEP engine

- The CEP service starts automatically when the operating system starts. The CEP engine that was running the last time the operating system shut down also starts automatically.

- If cepstopeng or cepstopserv has been used to stop the CEP engine and the operating system is then restarted, the CEP engine will not start automatically when the operating system starts. To start the CEP engine, execute cepstarteng again.

Example

Starting the service

When cepstartserv is used to start the CEP service.

```
# cepstartserv<ENTER>
(...)
Interstage Java EE DAS           started
(...)
Interstage Java EE Node Agent    started
(...)
UX:ismngconsolestart: INFO: is40041: The service has been activated normally.
(...)
UX:IHS: INFO: ihs01000: The command terminated normally.
(...)
LOG:  database system is ready to accept connections
(...)
Starting Tomcat:                                           [  OK  ]

Command cepstartserv executed successfully.
```

6.1.3 Deploying and Undeploying Definition Information

This section explains how to deploy the definition information stored on the CEP Server, and how to undeploy definition information no longer required.

6.1.3.1 Deploying Definition Information

Execute cepdeployrsc to deploy the definition information on the CEP Server.

General user permissions can be used to execute this command.

If the CEP engine is not stopped, then you must stop it (refer to "6.1.6 Stopping the CEP Engine" for details).

Example

Deploying definition information

When deploying the following definition information (definition file) to a CEP engine (CepEngine).

```
/application
    +-- EVENT01.xml      ... Event type definition    (development asset ID: EVENT01)
    +-- RULE01.xml       ... Rule definition          (development asset ID: RULE01)
    +-- MASTER01.xml     ... Master definition        (development asset ID: MASTER01)
    +-- RDBREF01.xml     ... RDB reference definition (development asset ID: RDBREF01)
    +-- LISTENER01.xml   ... SOAP listener definition (development asset ID: LISTENER01)
```

An example of command execution is shown below.

```
$ cepdeployrsc eventtype -e CepEngine -f /application/EVENT01.xml<ENTER>
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]: <ENTER>
Command cepdeployrsc executed successfully.
$ cepdeployrsc rule -e CepEngine -f /application/RULE01.xml<ENTER>
(...)
$ cepdeployrsc master -e CepEngine -f /application/MASTER01.xml<ENTER>
(...)
$ cepdeployrsc rdb_ref -e CepEngine -f /application/RDBREF01.xml<ENTER>
(...)
$ cepdeployrsc listener -e CepEngine -f /application/LISTENER01.xml<ENTER>
(...)
```

6.1.3.2 Checking Deployed Definition Information

Execute cepdispeng and cepgetrsc to check deployed definition information.

Execute cepdispeng with the `-i` option to list the definition information deployed in the CEP engine, and cepgetrsc to check details of the deployed definition information.

General user permissions can be used to execute these commands. The definition information can be checked regardless of whether the CEP engine is stopped or running.

Example

Listing the deployed definition information

When listing the definition information deployed to a CEP engine (CepEngine).

```
$ cepdispeng -i -e CepEngine<ENTER>
engineId         :CepEngine
eventtype        :EVENT01
rule             :RULE01
master           :MASTER01
rdb_ref          :RDBREF01
listener         :LISTENER01
Command cepdispeng executed successfully.
```

Listing details of the deployed definition information

When listing the details of an event type definition (EVENT01) deployed to a CEP engine (CepEngine).

```
$ cepgetrsc eventtype -e CepEngine -n EVENT01<ENTER>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventType xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="EVENT01">
    <comment>Event type definition</comment>
    <type>CSV</type>
    <xmlSchema></xmlSchema>
    <csvColumn>
        <column name="memberID" type="string"/>
        <column name="areaID" type="string"/>
        <column name="status" type="string"/>
    </csvColumn>
    <root></root>
    <useLogging>false</useLogging>
    <loggingTableName></loggingTableName>
    <useCep>true</useCep>
</eventType>

Command cepgetrsc executed successfully.
```

Note

- If redirecting the command output of the detailed display (XML format), modify each item accordingly.
- In addition to XML format data, the results of cepgetrsc also include output messages. If redirecting the command output to a file, delete unnecessary messages before using the results.

6.1.3.3 Updating Deployed Definition Information

Execute cepdeployrsc with the `-o` option to update the deployed definition information.

General user permissions can be used to execute this command.

If the CEP engine is not stopped, then you must stop it (refer to "6.1.6 Stopping the CEP Engine" for details).

Refer to "6.1.5.5 Dynamically Changing Rule Definitions and Master Data" to update rule definitions or the master data while the CEP engine is running.

Example

Updating the deployed definition information

When updating an event type definition (EVENT01) of a CEP engine (CepEngine) with new details.

The path of the event type definition with the new details (development asset ID: EVENT01) is shown below.

```
/application/EVENT01_new.xml
```

An example of command execution is shown below.

```
$ cepdeployrsc eventtype -o -e CepEngine -f /application/EVENT01_new.xml<ENTER>
Are you sure you want to deploy the event type definition?(default: y) [y,n,q]: <ENTER>
Command cepdeployrsc executed successfully.
```

6.1.3.4 Undeploying Definition Information

Execute cepundeployrsc to undeploy definition information no longer required.

General user permissions can be used to execute this command. Release the definition information while the CEP engine is stopped.

Refer to "6.1.6 Stopping the CEP Engine" if the CEP engine needs to be stopped.

Example

Undeploying the definition information

When undeploying the following definition information deployed to a CEP engine (CepEngine).

Definition information type	Development asset ID
Event type definition	EVENT01
Rule definition	RULE01
Master definition	MASTER01

Definition information type	Development asset ID
RDB reference definition	RDBREF01
SOAP listener definition	LISTENER01

An example of command execution is shown below.

```
$ cepundeployrsc eventtype -e CepEngine -n EVENT01<ENTER>
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]: <ENTER>
Command cepundeployrsc eventtype executed successfully.
$ cepundeployrsc rule -e CepEngine -n RULE01<ENTER>
(...)
$ cepundeployrsc master -e CepEngine -n MASTER01<ENTER>
(...)
$ cepundeployrsc rdb_ref -e CepEngine -n RDBREF01<ENTER>
(...)
$ cepundeployrsc listener -e CepEngine -n LISTENER01<ENTER>
(...)
```

6.1.4 Starting the CEP Engine

Execute cepstarteng to start all CEP engines to be used - starting will be performed one CEP engine at a time.

General user permissions can be used to execute this command.

Example

When the command is executed.

```
$ cepstarteng -e CepEngine<ENTER>
Command cepstarteng executed successfully.
```

Note

Starting the CEP engine will fail in the following cases:

- If there has been no deployment of at least one event type definition.
- If an XML schema specified in an event type definition contains an error.
- If the syntax of a filter rule or complex event processing rule specified in a rule definition contains an error.

Information

How to check the CEP engine name

Execute cepdispeng to list the CEP engines that have been set up.

6.1.5 Typical Operation Tasks

This section explains the following operation tasks performed after the CEP engine is started:

- Displaying the operation status of the CEP service
- Displaying the operation status of the CEP engine
- Monitoring abnormalities using logs
- Checking the resource usage of the CEP engine

- Dynamically changing rule definitions and master data

6.1.5.1 Displaying the Operation Status of the CEP Service

Execute cepdispserv as a superuser to display the status of the CEP service.

This allows you to check that the CEP service is running normally at any time, such as immediately after operation starts or during peak business hours. Execute cepstartserv as a superuser to start any service that is not running.

Example

Using cepdispserv to check the operation status of the CEP service

Check that the content below is output (*`nnnn`* indicates the process number, *`cepEngine`* indicates the created CEP engine name):

```
# cepdispserv<ENTER>
(...)
Interstage Java EE DAS          started
(...)
Interstage Java EE Node Agent   started
(...)
CEPAgentIJServerCluster running
cepEngine_flt not running
cepEngine_cep not running
(...)
Status            : Running
(...)
jsvc (pid nnnn nnnn) is running...
(...)
pg_ctl: server is running (PID: nnnn)
(...)
Command cepdispserv executed successfully.
```

Information

The CEP service entity

A CEP service is made up of multiple processes. Refer to "8.5 cepdispserv" for details.

6.1.5.2 Displaying the Operation Status of the CEP Engine

Execute cepdispeng to display the status of the CEP engine.

This allows you to check that the CEP engine is running normally at any time, such as immediately after operation starts or during peak business hours (refer to Chapter 5, "Errors during Operation" in *Troubleshooting* if the CEP engine has ABNORMAL or STOP status when it should be running).

General user permissions can be used to execute this command.

Example

Checking the operation status of the CEP engine

When checking the operation status of a CEP engine (CepEngine) - information such as the number of input and output events is also displayed.

```
$ cepdispeng -e CepEngine<ENTER>
engineId         :CepEngine
(...)
status_filter    :RUN
status_cep       :RUN
```

```
inEvent_filter   :100
inEvent_cep      :100
outEvent_filter  :100
outEvent_cep     :100
(...)
Command  cepdispeng executed successfully.
```

6.1.5.3 Monitoring Abnormalities Using Logs

When an abnormality occurs in the CEP service and the CEP engine, a message will be output to the system log and to the engine logs.

The system log and the engine logs are used differently. For example, in addition to the messages reporting abnormalities, the engine logs also contained more detailed information on CEP engine operation, such as records sent using the SOAP listener and debug information. Therefore, the usual abnormality monitoring monitors only the system log and, when an abnormality occurs, references the engine logs to analyze its cause.

When an abnormality occurs, respond on the basis of the messages output to each log. Refer to *Messages* for information on specific responses.

The details of log output are explained as follows:

- Output destinations of the system log and the engine logs

- Message format

Output destinations of the system log and the engine logs

The output destinations of the system log and the engine logs are shown below.

One CEP engine outputs two engine logs - one contains output relating to input adapter and high-speed filter processing, and the other contains output relating to complex event processing and output adapter processing. The engine logs have no predetermined format.

System log

```
/var/log/messages
```

Engine log for the high-speed filter

```
/var/opt/FJSVcep/cep/flt/logs/EngineLog/cepEngineName/engine.log
```

Engine log for complex event processing

```
/var/opt/FJSVcep/cep/cep/logs/EngineLog/cepEngineName/engine.log
```

The engine logs of the high-speed filter and complex event processing undergo rotation as follows:

- File size: 10 MB

- Rotation generations: 9 generations

Note

Engine logs older than 9 generations are automatically deleted. If they are required, periodically store them in a separate location.

Message format

The format of the messages output by BDCEP is shown below.

Refer to Section 1.1, "Message Format" in *Messages* for details.

```
CSPF_CEP: errorType: messageNumber: messageText
```

- Each element is delimited by a colon (`:`).

Example

Displaying a message

An example message is shown below.

```
CSPF_CEP: ERROR: cep10108e: Event type is not found. EngineId=CepEngine, eventType=EVENT01
```

6.1.5.4 Checking the Resource Usage of the CEP Engine

The resource usage of the CEP engine (such as the heap memory usage of Java VM) is recorded in the resource logs. The logs are output in CSV format and therefore can be analyzed using a tool such as Excel, in order to detect resource excesses or deficiencies in the CEP engine.

The details of the resource logs are explained as follows:

- Output destinations of the resource logs
- Format of the resource log for the high-speed filter
- Format of the resource log for complex event processing
- Collection interval for the resource usage

Output destinations of the resource logs

Two resource logs are output for each CEP engine - one contains output relating to input adapter and high-speed filter processing, and the other contains output relating to complex event processing and output adapter processing.

The output destination of each resource log is shown below.

Resource log for the high-speed filter

```
/var/opt/FJSVcep/cep/flt/logs/ResourceLog/cepEngineName/resource.log
```

Resource log for complex event processing

```
/var/opt/FJSVcep/cep/cep/logs/ResourceLog/cepEngineName/resource.log
```

The resource logs of the high-speed filter and complex event processing undergo rotation as follows:

- Point of rotation: One day's worth (specific point of rotation is set using cron)
- Rotation generations: 13 generations

Note

Resource logs older than 13 generations are automatically deleted. If they are required, periodically store them in a separate location.

Format of the resource log for the high-speed filter

This section explains the format of the resource log for the high-speed filter.

The format is shown below:

```
time,resourceId,jheapNewUsed,jheapNewFree,jheapNewTotal,jheapOldUsed,jheapOldFree,
jheapOldTotal,jheapNewPlusOldTotal,jheapPermUsed,jheapPermFree,jheapPermTotal,VSZ,
numOfInEvents,numOfOutEvents,numOfLogs,numOfRulesDeployed,numOfSocketConnections
```

Note that in the format above, newlines have been added for readability only. The actual log does not have newlines.

The table below explains each item:

Item name	Content
time	Output date and time of the resource log. The format is as follows: `yyyy-MM-dd HH:mm:ss` - *yyyy*: Year - *MM*: Month - *dd*: Day - *HH*: Hour - *mm*: Minute - *ss*: Second
resourceId	Resource ID. The format is as follows: `CSPF_CEP_hostName_engineId_flt`
jheapNewUsed	Java VM heap memory usage (new generation area) for the high-speed filter (unit: bytes) (*1)
jheapNewFree	Java VM heap free memory (new generation area) for the high-speed filter. (unit: bytes) (*1)
jheapNewTotal	Java VM heap memory (new generation area) for the high-speed filter. (unit: bytes) (*1)
jheapOldUsed	Java VM heap memory usage (old generation area) for the high-speed filter. (unit: bytes) (*1)
jheapOldFree	Java VM heap free memory (old generation area) for the high-speed filter. (unit: bytes) (*1)
jheapOldTotal	Java VM heap memory (old generation area) for the high-speed filter. (unit: bytes) (*1)
jheapNewPlusOldTotal	Java VM heap memory (new generation area + old generation area) for the high-speed filter. (unit: bytes) (*1)
jheapPermUsed	Java VM heap memory usage (permanent generation area) for the high-speed filter. (unit: bytes) (*1)
jheapPermFree	Java VM heap free memory (permanent generation area) for the high-speed filter. (unit: bytes) (*1)
jheapPermTotal	Java VM heap memory (permanent generation area) for the high-speed filter. (unit: bytes) (*1)
VSZ	Process memory usage for the high-speed filter. (unit: KB) (*1)
numOfInEvents	Number of events input from devices on which events are occurring. (*1)
numOfOutEvents	Number of events sent to the engine on the complex event processing side. (*1)
numOfLogs	Number of logs in the input adapter. (*1)
numOfRulesDeployed	Number of rules deployed to the high-speed filter (number of high-speed filter statements). (*1)
numOfSocketConnections	Number of simultaneous socket connections. (*1)

*1: If the CEP engine status is not RUN (running normally), an empty string ("") will be output.

Example

Content of the resource log (high-speed filter)

```
2012-08-01 14:00:04,CSPF_CEP_cepsv_CepEngine_flt,30130928,9125136,39256064,39224768,
475888192,515112960,554369024,68207736,17292168,85499904,3670240,0,0,0,2,0
```

Note that in the example above, a newline has been added for readability only. The actual log does not have newlines.

Format of the resource log for complex event processing

This section explains the format of the resource log for complex event processing.

The format is shown below:

```
time,resourceId,jheapNewUsed,jheapNewFree,jheapNewTotal,jheapOldUsed,jheapOldFree,
jheapOldTotal,jheapNewPlusOldTotal,jheapPermUsed,jheapPermFree,jheapPermTotal,VSZ,
reserve1,reserve2,reserve3,reserve4,reserve5,reserve6,numOfInEvents,numOfOutEvents,
numOfLogs,numOfRulesDeployed,numOfListenersDeployed
```

Note that in the format above, newlines have been added for readability only. The actual log does not have newlines.

The table below explains each item:

Item name	Content
time	Output date and time of the resource log. The format is as follows: `yyyy-MM-dd HH:mm:ss` - *yyyy*: Year - *MM*: Month - dd: Day - *HH*: Hour - *mm*: Minute - sis: Second
resourceId	Resource ID. The format is as follows: `CSPF_CEP_hostName_engineId_cep`
jheapNewUsed	Java VM heap memory usage (new generation area) for complex event processing. (unit: bytes) (*1)
jheapNewFree	Java VM heap free memory (new generation area) for complex event processing. (unit: bytes) (*1)
jheapNewTotal	Java VM heap memory (new generation area) for complex event processing. (unit: bytes) (*1)
jheapOldUsed	Java VM heap memory usage (old generation area) for complex event processing. (unit: bytes) (*1)
jheapOldFree	Java VM heap free memory (old generation area) for complex event processing. (unit: bytes) (*1)
jheapOldTotal	Java VM heap memory (old generation area) for complex event processing. (unit: bytes) (*1)
jheapNewPlusOldTotal	Java VM heap memory (new generation area + old generation area) for complex event processing. (unit: bytes) (*1)
jheapPermUsed	Java VM heap memory usage (permanent generation area) for complex event processing. (unit: bytes) (*1)
jheapPermFree	Java VM heap free memory (permanent generation area) for complex event processing. (unit: bytes) (*1)

Item name	Content
jheapPermTotal	Java VM heap memory (permanent generation area) for complex event processing. (unit: bytes) (*1)
VSZ	Process memory usage for complex event processing. (Unit: KB) (*1)
reserve1	Reserved areas. Empty strings ("") will be output for all of these.
reserve2	
reserve3	
reserve4	
reserve5	
reserve6	
numOfInEvents	Number of events input from high-speed filter processing. (*1)
numOfOutEvents	Number of events sent to user-developed Web services. (*1)
numOfLogs	Number of logs in complex event processing. (*1)
numOfRulesDeployed	Number of rules deployed to complex event processing (number of complex event processing statements). (*1)
numOfListenersDeployed	Number of deployed SOAP listener definitions. (*1)

*1: If the CEP engine status is not RUN (running normally), an empty string ("") will be output.

Example

Content of the resource log (complex event processing)

```
2012-08-01 14:00:04,CSPF_CEP_cepsv_CepEngine_cep,14042608,25213456,39256064,41802024,
473310936,515112960,554369024,76981880,8481160,85463040,3385020,,,,,,,0,0,0,6,0
```

Note that in the example above, a newline has been added for readability only. The actual log does not have newlines.

Collection interval for resource usage

The resource usage is collected in 10-minutes intervals and is output to the resource log.

This process is performed by using cron to periodically call an obtain process.

The cron setting is set in the following file. The engine execution user is the user name specified at installation:

```
/var/spool/cron/engineExecutionUser
```

Note

Do not change the contents of the above cron setup file.

6.1.5.5 Dynamically Changing Rule Definitions and Master Data

You can change (using dynamic change) the rule definitions and the master data of a CEP engine while it is running without stopping it. Doing this simplifies the process for maintaining rule definitions and master data, because the CEP engine need not be restarted.

To dynamically change rule definitions and the master data, use the -h option of cepdeployrsc.

Refer to "8.3 cepdeployrsc" for information on the cepdeployrsc command.

Users with general user permissions can execute this command. Dynamic change of rule definitions and the master data is performed while the CEP engine is running.

Note

If there is a change in a complex event processing rule when a rule definition is dynamically changed, the information held in a window by the complex event processing rule is lost. Refer to "8.3 cepdeployrsc" for additional notes on dynamic changes.

Example

Dynamically changing a rule definition

When dynamically changing a rule definition (RULE01) of a CEP engine (CepEngine) with new details.

The path of the rule definition with the new details (development asset ID: RULE01) is shown below.

```
/tmp/RULE01_new.xml
```

An example of command execution is shown below.

```
$ cepdeployrsc rule -h -e CepEngine -f /tmp/RULE01_new.xml<ENTER>
Are you sure you want to hotdeploy the rule definition?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
```

Dynamically changing master data

When dynamically changing the contents of a data file specified in a master definition deployed to a CEP engine (CepEngine).

```
$ cepdeployrsc master -h -e CepEngine<ENTER>
Are you sure you want to hotdeploy the master data?(default: y) [y,n,q]:<ENTER>
Command cepdeployrsc executed successfully.
```

6.1.5.6 Storing the Custom Log

If a user-developed Java class is used, it may output logs to the custom log. The output destination of the custom log is shown below.

```
/var/opt/FJSVcep/cep/cep/logs/EnginLog/engineName/custom.log
```

The custom log is rotated as shown below. Old custom logs are automatically deleted. Therefore, if you need to retain old custom logs, periodically store them in a separate location.

- File size: 10 MB
- Rotation generations: 9 generations

6.1.6 Stopping the CEP Engine

Execute cepstopeng to stop the CEP engine.

General user permissions can be used to execute this command.

Example

When a CEP engine (CepEngine) is stopped.

```
$ cepstopeng -e CepEngine<ENTER>
Command cepstopeng executed successfully.
```

Point

When cepstopserv is used, all CEP engines running will also be stopped.

6.1.7 Stopping the CEP Service

Execute cepstopserv as a superuser to stop the CEP service. Note that the command also stops all CEP engines running.

Example

When the command is executed.

```
# cepstopserv<ENTER>
Stopping Engines.
(...)
Command cepstopeng executed successfully.
(...)
Shutting down Tomcat:                                      [  OK  ]
(...)
server stopped
(...)
UX:IHS: INFO: ihs01000: The command terminated normally.
(...)
UX:ismngconsolestop: INFO: is40042: The service has been terminated normally.
(...)
Interstage Java EE Node Agent    stopped
(...)
Interstage Java EE DAS           stopped
(...)
UX:isstop: INFO: is30160: INTERSTAGE terminated normally.

Command cepstopserv executed successfully.
```

Information

When the operating system shuts down, the CEP service stops automatically.

6.1.8 Stopping the Collaboration System

If Terracotta collaboration, RDB collaboration, or Hadoop collaboration is being performed, stop the relevant collaborating system after stopping the CEP Server.

Refer to the manual for the relevant product for information on how to stop the system.

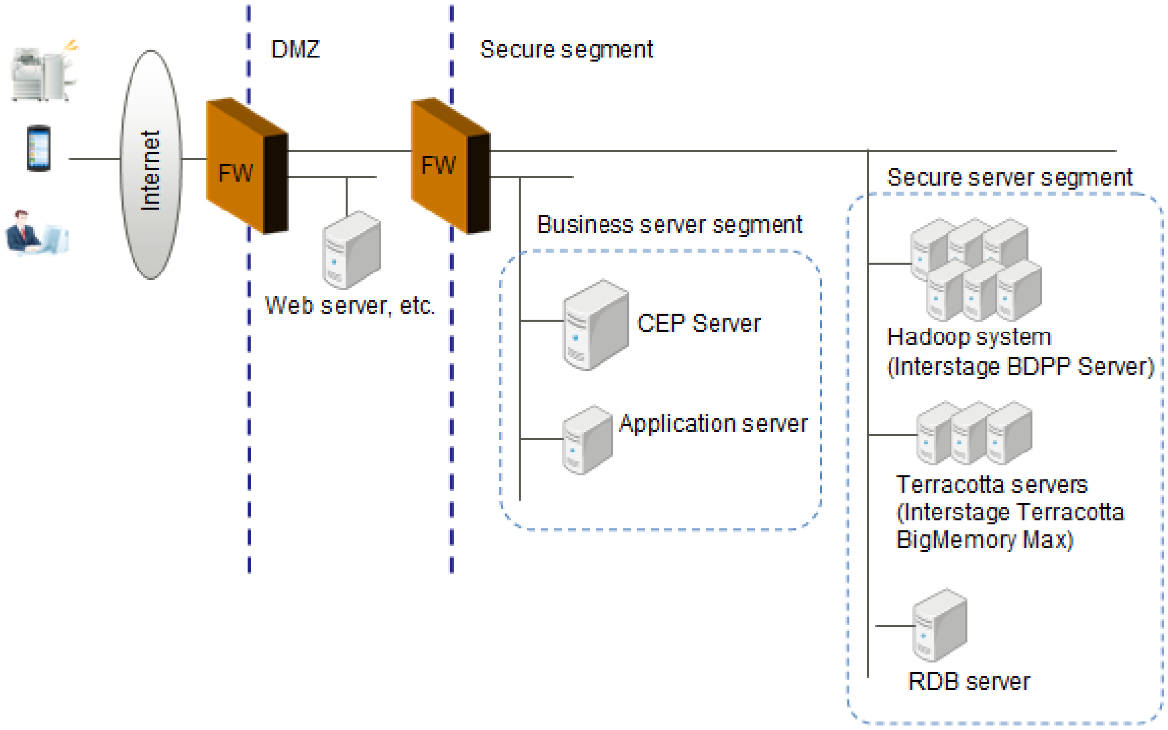
6.2 Security

This section provides information required to operate BDCEP securely.

6.2.1 Operation Model

A typical operation model of BDCEP is shown below.

Figure 6.1 Typical operation model of BDCEP



6.2.2 Prerequisite Knowledge for Designing Security

Take the following elements into consideration when designing, in order to achieve secure system operations using BDCEP:

- Security roles
- Protected resources
- Threats to protected resources and their countermeasures
- Overview of countermeasures for threats to protected resources

Security roles

The table below lists the types of system users that use BDCEP, their security roles, and the corresponding operating system user:

User type	Security role	Operating system user
System administrator	Can perform all operations. Can perform operations involving CEP Server, such as starting and stopping the CEP Server or reconfiguring a CEP engine.	Superuser
Engine execution user	Can run a CEP engine process.	Engine execution user (create at installation)
Developer	Can deploy definition information to a CEP engine and undeploy it (to check the operation of definition information), as well as start and stop a CEP engine.	General user who can login to the CEP Server

Protected resources

The table below lists the resources to be protected by the CEP Server.

Type	Protected resource	Description
File	Engine configuration file	File used to reconfigure a CEP engine.
	Deployed definition information	Definition information deployed to a CEP engine.
	Master data	CSV files to be referenced by the high-speed filter.
	Resource log	Output file used to investigate the resource usage.
	Engine log	File to which the detailed operation status of a CEP engine is output.
	Custom log	Log file output by a user-developed Java class
	Data for investigation	File collected to investigate faults.
Network	Event data sent to an input adapter	Packets traveling across the network.
	SOAP messages sent from an output adapter	
	Communication when Terracotta collaboration is used to remotely access a cache	
	Communication when Hadoop collaboration is used to remotely access a Hadoop system	
	Communication when RDB collaboration is used to remotely access a relational database	

Threats to protected resources and their countermeasures

The table below lists the possible threats to protected resources, and their respective countermeasures:

Type of protected resource	Threat	Security countermeasure
File	Tampering or destroying	Setting permissions Authenticating operation permissions for the CEP Server
Network	Sniffing	Placing on a secure segment

Overview of countermeasures for threats to protected resources

The table below provides an overview of each possible security countermeasure:

Security countermeasure	Overview of countermeasure
Setting permissions	Set operating system permissions for files included in the protected resources. Set appropriate permissions to suit the security roles.
Authenticating operation permissions for the CEP Server	Use operating system authentication. Only allow suitable users to login to the operating system. BDCEP assumes that users who are given authentication to login to the CEP Server can be trusted with operating a CEP engine and referencing an engine log. In addition, some operations, such as reconfiguring a CEP engine and starting and stopping the CEP service, can only be performed by a superuser.
Placing on a secure segment	To inhibit data sniffing and hacking, place the CEP Server on a secure segment.

6.2.3 Designing Security for BDCEP

This section explains how to design security for the systems using BDCEP, as follows:

- Authenticating for the CEP Server
- Designing suitable access permissions
- Designing the network

Authenticating for the CEP Server

A superuser of the operating system of the server to which BDCEP is applied can operate the CEP Server.

In addition, authentication is not performed when events are sent to the CEP Server from outside the system. Build a firewall or use event sender business applications to build a system in which authentication is performed when events are sent to the CEP Server.

Designing suitable access permissions

Set suitable access permissions for the files below as a countermeasure to prevent file tampering and destruction.

The table below describes the access permissions to be set for each file.

File	Reference permission	Write permission
Engine configuration file	Superuser	Superuser
Master data	Engine execution user	None
Data for investigation	Superuser	Superuser

The access permissions for files generated by a CEP engine, such as the event log and resource log, will be set automatically.

Designing the network

If the system has been located according to the system configuration supported by BDCEP, a third party will be unable to reference data transmitted over the network.

6.3 Maintenance

This section explains the operations required when maintaining BDCEP.

6.3.1 Collecting Data for Investigation when a Problem Occurs

BDCEP provides a command to collect data for investigation when a problem occurs (refer to Chapter 1, "Collecting Diagnostics Data" in *Troubleshooting* for details).

6.3.2 Backup and Restore

Resources must be backed up periodically, in case the system on the CEP Server fails.

The table below lists the resources to be backed up.

Resource name	Content	When to back up
Engine configuration file	File defining the CEP engine configuration	When settings are changed.
Master data	Master data to be stored in the CEP engine	When storing on the CEP Server. When editing on the CEP Server.
Definition information (definition file)	Definition information to be deployed to the CEP engine	When storing on the CEP Server. When editing on the CEP Server.
User-developed Java class	jar file or class file of the user-developed Java class	When storing on the CEP Server.
Setup files for Terracotta collaboration	Files required for Terracotta collaboration	When changing settings

Resource name	Content	When to back up
Setup file for RDB collaboration	File required for RDB collaboration	When changing settings

- The event log stored on the CEP Server is for checking operation, so it does not need to be backed up or restored.
- If the master data is backed up by the data provider, it does not need be backed up.
- If the definition information is backed up by the definition information developer, it does not need to be backed up.
- If the user-developed Java class has been backed up by the developer, it need not be backed up.
- Refer to the manual of the relevant collaboration product for information on the backup and restore of resources in the collaboration system.

6.3.2.1 Backup Procedure

Follow the steps below as a superuser to back up the resources:

1. Back up the engine configuration file
2. Back up the master data
3. Back up the definition information (definition file)
4. Back up the user-developed Java class
5. Back up the setup files for Terracotta collaboration
6. Back up the setup file for RDB collaboration

This explanation uses the following directory to back up the resources:

```
/backup
```

Back up the engine configuration file

Back up the engine configuration file specified during execution of cepconfigeng.

Example

When the engine configuration file is "`/etc/opt/FJSVcep/Engine.xml`".

```
# cp -p /etc/opt/FJSVcep/Engine.xml /backup/<ENTER>
```

Back up the master data

Back up the master data stored on the CEP Server.

Example

When the master data is stored in "`/masterdata`".

```
# cp -rp /masterdata /backup/<ENTER>
```

Back up the definition information (definition file)

Back up the definition information (definition file) stored on the CEP Server.

Example

When the definition file is stored in "`/application`".

```
# cp -rp /application /backup/<ENTER>
```

Note

RDB reference definition file

The RDB reference definition file contains the password, so for security reasons you must take measures before backing up the file to ensure that the password cannot be referenced. Do this by deleting the access password from the definition file, or encrypting the entire definition file.

Back up the user-developed Java class

Back up /etc/opt/FJSVcep/config/custom.

Example

When backing up the definition information of the user-developed Java class.

```
# cp -rp /etc/opt/FJSVcep/config/custom /backup/ <ENTER>
```

Back up the setup files for Terracotta collaboration

Back up /etc/opt/FJSVcep/config/ehcache.xml and /etc/opt/FJSVcep/config/terracotta_conf.

Example

When backing up the definition information of the setup files for Terracotta collaboration.

```
# cp -p /etc/opt/FJSVcep/config/ehcache.xml /backup/<ENTER>
# cp -p /etc/opt/FJSVcep/config/terracotta_conf /backup/<ENTER>
```

Back up the setup file for RDB collaboration

Back up /etc/opt/FJSVcep/config/rdb_conf.

Example

When backing up the definition information of the setup file for RDB collaboration.

```
# cp -p /etc/opt/FJSVcep/config/rdb_conf /backup/<ENTER>
```

6.3.2.2 Restore Procedure

Follow the steps below as a superuser to restore the resources:

1. Restore the engine configuration file

2. Restore the master data

3. Restore the definition information

4. Restore the user-developed Java class

5. Restore the setup files for Terracotta collaboration

6. Restore the setup file for RDB collaboration

This section assumes that BDCEP has been installed and set up (refer to "Chapter 4 Installation and Setup" for details).

It also assumes that the following resource backup directory exists:

```
/backup
```

Restore the engine configuration file

Restore the engine configuration file (refer to "Chapter 4 Installation and Setup" for information on how to reconfigure a CEP engine using a restored engine configuration file).

Example

When the engine configuration file that is the backup source is in "`/etc/opt/FJSVcep/Engine.xml`" and the backed up engine configuration file is "`/backup/Engine.xml`".

```
# cp -p /backup/Engine.xml /etc/opt/FJSVcep/<ENTER>
```

Restore the master data

Restore the master data.

Example

When the master data that is the backup source is in "`/masterdata`" and the backed up master data is "`/backup/masterdata`".

```
# cp -rp /backup/masterdata /<ENTER>
```

Restore the definition information (definition file)

Restore the definition information (definition file) (refer to "6.1.3.1 Deploying Definition Information" for information on how to redeploy restored definition information).

Example

When the definition information (definition file) that is the backup source is in "`/application`" and the backed up definition information is "`/backup/application`".

```
# cp -rp /backup/application /<ENTER>
```

Restore the user-developed Java class

Restore the user-developed Java class.

Example

When the backed up user-developed Java class is stored in "/backup/custom".

```
# cp -rp /backup/custom /etc/opt/FJSVcep/config/ <ENTER>
```

Restore the setup files for Terracotta collaboration

Restore the setup files for Terracotta collaboration.

Example

When the backed up setup file is "/backup/ehcache.xml" or "/backup/terracotta_conf".

```
# cp -p /backup/ehcache.xml /etc/opt/FJSVcep/config/ <ENTER>
# cp -p /backup/terracotta_conf /etc/opt/FJSVcep/config/ <ENTER>
```

Restore the setup file for RDB collaboration

Restore the setup file for RDB collaboration.

Example

When the backed up setup file is "/backup/rdb_conf".

```
# cp -p /backup/rdb_conf /etc/opt/FJSVcep/config/ <ENTER>
```

6.3.3 Applying Updates

Update information provided by software products including BDCEP can be obtained from "UpdateSite", which is an integrated software update information provision site.

For BDCEP, apply the updates released from the products below:

Product name	Version
Interstage Big Data Complex Event Processing Server	V1.1.0
Interstage Application Server Enterprise Edition (64-bit)	V11.1.0

Follow the steps below as a superuser to apply an update to BDCEP:

1. Stop cron.

   ```
   # service crond stop<ENTER>
   ```

2. Execute cepstopserv.

   ```
   # cepstopserv<ENTER>
   ```

3. Apply the update.

   Apply the update according to the update information file.

4. If Terracotta collaboration has been set up and Interstage Application Server Enterprise Edition (64 bit) updates have been applied, the files below saved by the procedure in "4.4.3 Setup of Terracotta Collaboration" may be restored. In this case, re-execute the procedure for modifying the names of the restored files.

   - /opt/FJSVisjee/lib/jersey-bundle-1.0.3.1.jar

   - /opt/FJSVisjee/lib/jsr311-api-1.0.jar

   - /opt/FJSVisjee/lib/jettison-1.0.1.jar

   - /opt/FJSVisjee/lib/jackson-asl-0.9.4.jar

5. Execute cepstartserv.

   ```
   # cepstartserv<ENTER>
   ```

6. Start cron.

   ```
   # service crond start<ENTER>
   ```

Note

Note on when cron is stopped

BDCEP uses cron to obtain the resource usage, which is configured by BDCEP using the crontab of the engine execution user.

When the CEP service is stopped, and when the process to obtain the resource usage from the cron service is executed, an error may be output using the execution cycle set in the cron service (for obtaining the resource usage, 10-minutes intervals). This error is canceled by starting the CEP service again.

When cron is stopped, both the cron setting for the process to obtain the resource usage and the cron setting for other than the engine execution user stop. Therefore, when applying updates, select a time period when stopping cron will not cause problems.

If the manual is also being updated according to the update information file attached to the update, obtain the latest manual from the following site:

```
http://www.fujitsu.com/support/software/manual/ (as at December 2013)
```

6.3.4 Tuning

This section explains tuning for BDCEP, as follows:

- Tuning JVM options
- Tuning file descriptors
- Tuning trace logs

6.3.4.1 Tuning JVM Options

Two Java VMs (hereafter, referred to as "JVMs") operate within one CEP engine - one performs input adapter and high-speed filter processing, and the other performs complex event processing and output adapter processing.

If analysis of the resource log has revealed an excess or insufficiency of JVM heap memory, tune the JVM options of the CEP engine. This will avoid problems such as decline in performance caused by garbage collections in the CEP engine.

The table below lists the JVM options that can be tuned:

JVM option	Initial value at CEP engine creation
Maximum value of the memory allocation pool	2048MB
Initial value of the memory allocation pool	512MB
Maximum value of the permanent generation area	192MB

Follow the steps below to tune the JVM options:

1. Check the current JVM option settings.
2. Using the resource log, check the heap memory usage.
3. Calculate the heap memory size required - if there is enough memory, there is no need to tune.
4. Change the JVM option settings.
5. Start or restart the CEP engine (to reflect the changed settings).
6. Follow these steps again to check the changed settings.

Check the current JVM option settings

Execute cepgetjvmopt to check the JVM option settings of the CEP engine.

General user permissions can be used to execute this command.

Example

Executing cepgetjvmopt

When checking the JVM options of complex event processing for a CEP engine (CepEngine).

```
$ cepgetjvmopt cep -e CepEngine<ENTER>
xmxSize               :5120m
xmsSize               :256m
maxPermSize           :96m
Command cepgetjvmopt executed successfully.
```

Note that "m" stands for "MB".

Using the resource log, check the heap memory usage

Analyze the resource log to check the maximum value of the Java VM heap memory usage for the CEP engine. Specifically, check the new generation area, old generation area, and permanent generation area.

The following table provides items output by the resource log that are related to JVM heap memory.

This applies to the resource log of the high-speed filter and the resource log of complex event processing.

Refer to "6.1.5.4 Checking the Resource Usage of the CEP Engine" for information on the format of the resource log.

Heap memory-related item	Description
jheapNewUsed	Java VM heap memory usage (new generation area) for the high-speed filter.
jheapNewFree	Java VM heap free memory (new generation area) for the high-speed filter.
jheapNewTotal	Java VM heap memory (new generation area) for the high-speed filter.
jheapOldUsed	Java VM heap memory usage (old generation area) for the high-speed filter.
jheapOldFree	Java VM heap free memory (old generation area) for the high-speed filter.
jheapOldTotal	Java VM heap memory (old generation area) for the high-speed filter.
jheapNewPlusOldTotal	Java VM heap memory (new generation area + old generation area) for the high-speed filter.
jheapPermUsed	Java VM heap memory usage (permanent generation area) for the high-speed filter.
jheapPermFree	Java VM heap free memory (permanent generation area) for the high-speed filter.
jheapPermTotal	Java VM heap memory (permanent generation area) for the high-speed filter.

Calculate the heap memory size required

If the resource log has been used to analyze the respective maximum values of the new generation area, old generation area, and permanent generation area, calculate the heap memory size required, as shown below.

If the heap memory size required is less than the setting value, then there is no need to tune.

Calculate the maximum value of the memory allocation pool

```
(maxValOfNewGenerationArea + maxValOfOldGenerationArea + maxValOfPermanentGenerationArea) x 1.2
```

1.2 is the safety factor - if heap memory usage can vary widely with the time of the year or time period, set a higher safety factor.

Calculate the maximum value of the permanent generation area

```
maxValOfPermanentGenerationArea x 1.2
```

1.2 is the safety factor - if heap memory usage can vary widely with the time of the year or time period, set a higher safety factor.

Change the JVM option settings

Execute cepsetjvmopt as a superuser to change the JVM option settings of the CEP engine.

Example

Executing the cepsetjvmopt command

When changing the maximum value of the memory allocation pool in the JVM options for complex event processing of a CEP engine (CepEngine) to 512 MB. The respective defaults are used for the initial value of the memory allocation pool and the maximum value of the permanent generation area.

```
# cepsetjvmopt cep -xmx 512m -e CepEngine<ENTER>
Command cepsetjvmopt executed successfully.
```

Note that "m" stands for "MB".

Note

If an option of cepsetjvmopt (-xmx, -xms, or -xxmp) is omitted, the default value for that option will be used.

Start or restart the CEP engine.

Start or restart the CEP engine in order to reflect the changed JVM option settings in the CEP engine.

If the CEP engine is running, it must be stopped (refer to "6.1.6 Stopping the CEP Engine" for details).

Refer to "6.1.4 Starting the CEP Engine" for details.

General user permissions can be used to perform this task.

Example

Restarting the CEP engine

When restarting a running CEP engine (CepEngine).

```
$ cepstopeng -e CepEngine<ENTER>
Command cepstopeng executed successfully.
$ cepstarteng -e CepEngine<ENTER>
Command cepstarteng executed successfully.
```

6.3.4.2 Tuning File Descriptors

The number of file descriptors required for operating one CEP engine depends on the number of simultaneous socket connections or high-speed filter statements described in the rule definition filter rules.

If an error occurs due to insufficient filter descriptors, or file descriptors are found to be excessive or inadequate according to resource log analysis, it is necessary to specify the file descriptor upper limit.

The required number of file descriptors for each CEP engine is calculated as follows:

```
numOfFileDescriptorsRequired = A + (8 x B) + 371
```

Table 6.1 Placeholders in the file descriptor number tuning formula

Item	Description	Required Number
A	Number of simultaneous socket connections	Use the largest number of assumed simultaneous connections. Refer to the CEP engine operation status or the resource log output of the number of simultaneous connections to check the change in the current or previous number of simultaneous connections. Refer to "8.4 cepdispeng" or "6.1.5.4 Checking the Resource Usage of the CEP Engine" for information on how to reference it.
B	Total number of IF-THEN statements described in the filter rules	Refer to the CEP engine operation status or the resource log output of the number of high-speed filter statements to check the current number of IF-THEN statements. Refer to "8.4 cepdispeng" or "6.1.5.4 Checking the Resource Usage of the CEP Engine" for information on how to reference it.

Note

When the file descriptors are exhausted, operations might no longer be able to continue. Therefore, specify a value with some allowance for the number of simultaneous connections in *A*.

Compare the value calculated above with the actual number of file descriptors that can be used by the engine execution user. If there are multiple CEP engines, compare the largest value of the calculation result from each CEP engine.

Execute the command below as a superuser to display the maximum number of file descriptors that the engine execution user can use:

```
# /bin/su -c 'ulimit -n' engineExecutionUserName <ENTER>
```

If the calculated value is larger, then change `/etc/security/limits.conf` with it and restart the OS (refer to OS documentation for information on how to change the value).

Example

Setting /etc/security/limits.conf

This example changes the maximum number of file descriptors that the engine execution user "isbdcep" can use from the default value of 1024 to 2048.

```
isbdcep    soft    nofile    2048
isbdcep    hard    nofile    2048
```

6.3.4.3 Tuning Trace Logs

In BDCEP, the input-output information for event sender applications and receipt processing states for event sender application requests are output to trace logs.

By stopping the output of trace logs, it is possible to improve the efficiency of event processing when the HTTP adapter and SOAP adapter are used.

Note

Stopping the output of trace logs will mean that more time will have to be spent to determine the cause of eventual problems. If efficiency is not a problem, it is recommended to output the trace logs.

The flow to stop (or resume) trace log output is as follows (perform these operations as superuser):

(1) Stop (or Resume) Logging Event Sender Application I/O Information

(2) Stop (or Resume) Receipt Processing State of Event Sender Application Requests Log Output

(3) Restarting the CEP Service

(4) Starting the CEP Engine

(1) Stop (or Resume) Logging Event Sender Application I/O Information

Edit `/var/opt/FJSVihs/servers/FJapache/conf/httpd.conf` and set the `IHSTraceLog` setting value to "`off`".

Example

Setting /var/opt/FJSVihs/servers/FJapache/conf/httpd.conf

Before changing

```
IHSTraceLog "|/opt/FJSVihs/bin/ihsrlog -s logs/tracelog 2 5"
```

After changing

```
#IHSTraceLog "|/opt/FJSVihs/bin/ihsrlog -s logs/tracelog 2 5"
IHSTraceLog off
```

When resuming log output, also edit the `httpd.conf` file and revert the contents to its previous state.

(2) Stop (or Resume) Receipt Processing State of Event Sender Application Requests Log Output

Execute the commands below to stop the receipt processing state log output contents of the event sender application request. Note that in the command line example below, a backslash ("\") and newline have been added for readability only. The actual command line does not have backslash or newlines.

```
# /opt/FJSVisjee/bin/asadmin set \
 cepEngineName_flt-config.http-service.property.ISJEELogHttpTraceEnable=false <ENTER>
```

To resume log output, perform the following:

```
# /opt/FJSVisjee/bin/asadmin set \
 cepEngineName_flt-config.http-service.property.ISJEELogHttpTraceEnable=true <ENTER>
```

Example

`Executing the command`

If the CEP engine name is "`CepEngine`", perform the following to deter log output. Note that in the example below, a backslash ("\") and newline have been added for readability only. The actual command line does not have backslash or newline.

```
# /opt/FJSVisjee/bin/asadmin set \
  CepEngine_flt-config.http-service.property.ISJEELogHttpTraceEnable=false <ENTER>
CepEngine_flt-config.http-service.property.ISJEELogHttpTraceEnable=false
```

(3) Restarting the CEP Service

To reflect your changes, execute cepstopserv and then cepstartserv to restart the CEP service for the log output.

Example

Restarting the CEP service

```
# cepstopserv <ENTER>
(...)
```

```
# cepstartserv <ENTER>
(...)
```

(4) Starting the CEP Engine

Execute cepstarteng to start the CEP engine.

Example

Executing cepstarteng

```
# cepstarteng -e CepEngine <ENTER>
Command cepstarteng executed successfully.
```

Chapter 7 Extended System Operations

This chapter explains how to operate Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP") on multiple servers.

7.1 Scalable System Operations

This section explains scalable BDCEP operations using Terracotta Collaboration.

7.1.1 Scaleout of Complex Event Processing

If individual input events are simply to be filtered and output, you can install BDCEP on multiple servers and distribute input events using the front-end load balancer for processing them.

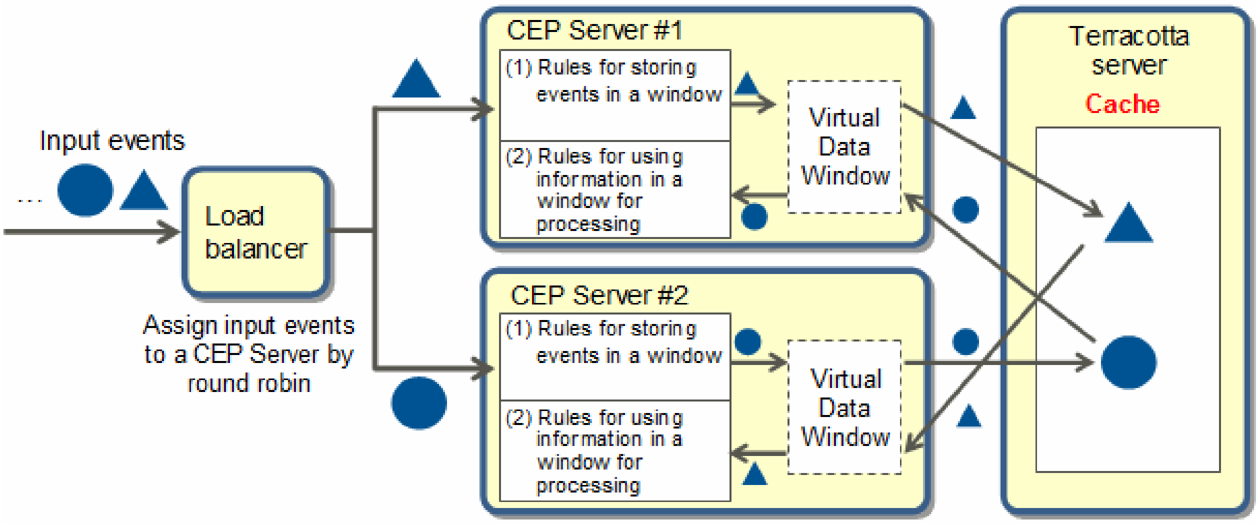
In some cases processing may not simply entail filtering but may instead specify a view, named window, OUTPUT clause, pattern expression, and aggregate function in the complex event processing rule. The input event information and the information generated as a result of processing the input event information are held in the memory. The information held in memory is then used to perform subsequent processing. In such cases, if a load balancer is used to simply distribute input events to multiple servers for processing, the information used for subsequent processing may not exist in the memory of the same server, and the anticipated output may not be achieved.

7.1.2 Scaleout Using Terracotta Collaboration

Terracotta collaboration provided by BDCEP enables the input event information and the information generated as a result of processing the input event information to be stored and referenced in the cache of an external Terracotta server. The cache of the Terracotta server can be accessed from multiple CEP servers.

You can operate BDCEP in a scaleout configuration where input events are distributed by a load balancer at the front-end of multiple CEP servers, and the cache on the same Terracotta server is referenced. To do this, use a Virtual Data Window to store the input event information and the information generated by processing the input event information in the cache of the Terracotta server, and then specify in a complex event processing rule that a Virtual Data Window is to be used to reference the stored information.

Figure 7.1 Example of referencing an event distributed to a different CEP Server by Terracotta collaboration



Input events
...
Load balancer
Assign input events to a CEP Server by round robin
CEP Server #1
(1) Rules for storing events in a window
(2) Rules for using information in a window for processing
Virtual Data Window
CEP Server #2
(1) Rules for storing events in a window
(2) Rules for using information in a window for processing
Virtual Data Window
Terracotta server
Cache

Note

- When using a Virtual Data Window to reference the Terracotta cache, you must use a join that specifies UNIDIRECTIONAL, ON SELECT statement, or subquery. Refer to "5.4.4.3.3 Using Terracotta cache" for details.

- Data in the cache of the Terracotta server is managed as entries, with each entry made up of a key and a value. Only one entry is held for the value of a property specified as a key. You must take this into account when designing rules.
- The process of referencing a Virtual Data Window for updating the data (value) in an entry is not a transaction process, so while one server is being used to reference and update data, a different server can perform an update.

7.2 Operating a Highly Reliable System Using PRIMECLUSTER

This section explains how to operate BDCEP in a reliable configuration using the highly reliable PRIMECLUSTER infrastructure software.

7.2.1 Overview of Reliable System Operations

The Cluster Service of BDCEP can be used to build a reliable system using PRIMECLUSTER, in order to prevent a long-term suspension of business due to hardware faults in the CEP Server.

This feature can be used to allow the events that occur after a cluster switch has completed to be processed by a CEP Server on the standby node.

Knowledge of PRIMECLUSTER is required to use the Cluster Service (refer to the PRIMECLUSTER manuals for details).

The range of application of the Cluster Service is shown below:

Operation form of a cluster service

1:1 standby operation

Cluster products that can be used

- PRIMECLUSTER Enterprise Edition 4.3A10 or higher
- PRIMECLUSTER HA Server 4.3A10 or higher

Features that can be used in a cluster system

All BDCEP features can be used.

Note

- The statuses of high-speed filter rules and complex event processing rules that exist on the active node prior to or during a cluster switch are not transferred to the CEP Server on the standby node.
- Events that occur during a switch are not received.

7.2.2 Cluster Service Configuration

To operate BDCEP in a cluster system, the CEP engine must be configured and the development assets (such as definition information and master data) must be deployed so that the respective servers on the active node and standby node will have the same resource configuration.

Set up PRIMECLUSTER so that the operating system also operates on the active node and the standby node using the same IP address, through IP address takeover.

There is no need to deploy BDCEP resources such as the definitions and engine log to a shared disk - instead, use the local disks of the active node and standby node.

7.2.3 Building a Cluster Service Environment

Follow the steps below to set up the environment to run BDCEP in a cluster system (refer to the PRIMECLUSTER manuals for information on installing and operating PRIMECLUSTER):

1. **Install and set up PRIMECLUSTER.**

   Install PRIMECLUSTER for both the active node and the standby node.

2. **Set up IP address takeover.**

   Set up PRIMECLUSTER so that the active node and standby node can use IP address takeover to switch and then operate using the same IP address.

3. **Install and set up BDCEP in the active node.**

   Activate IP address takeover in the active node, and then install and set up (for example, create a CEP engine) BDCEP, as usual (refer to "Chapter 4 Installation and Setup" for details).

   Note that you must register the active node server information in the master server connection authorization list when performing Hadoop collaboration (refer to Section 6.4, "Installing to a Development Server" in the *User's Guide* of the Interstage Big Data Parallel Processing Server V1.0.1 for details).

4. **Deploy development assets to the active node.**

   Deploy the required development assets, such as rule definitions, to the active node (refer to "5.8 Deploying Development Assets" for details).

   When a status in which events can be processed is reached, stop the CEP engine and the CEP service of the active node.

5. **Save the RC procedures of the active node.**

   Save the start shell script "S99startis" stored in the directories below to any directory for storing backup resources:

   - /etc/rc2.d
   - /etc/rc3.d
   - /etc/rc4.d
   - /etc/rc5.d

   Save the stop shell script "K00stopis" stored in the directories below to any directory for storing backup resources:

   - /etc/rc.d/rc0.d
   - /etc/rc.d/rc1.d
   - /etc/rc.d/rc6.d

   Disable the automatic start of the Interstage Java EE DAS service:

   ```
   # /sbin/chkconfig --del FJSVijdas <ENTER>
   ```

6. **Switch to the standby node.**

   Use a PRIMECLUSTER cluster switch operation to switch to the standby node.

7. **Install and set up BDCEP in the standby node.**

   With IP address takeover activated in the standby node, install and set up BDCEP in it. Make the CEP engine settings the same as those in the active node (refer to "Chapter 4 Installation and Setup" for details).

   Note that you must register the standby node server information into the master server connection authorization list when performing Hadoop collaboration (refer to Section 6.4, "Installing to a Development Server" in the *User's Guide* of the Interstage Big Data Parallel Processing Server V1.0.1 for details).

8. **Deploy development assets to the standby node.**

   Deploy the same development assets as those in the active node to the standby node (refer to "5.8 Deploying Development Assets" for details).

   There is no need to perform work that duplicates that in step 4, "Deploy development assets to the active node", such as providing data and deploying collaboration applications. Note, however, that if master data is being placed in the local disk of the CEP Server, it must be stored at the standby node in the same way as it was stored at the active node.

   When a status in which events can be processed is reached, stop the CEP engine and the CEP service of the standby node.

9. **Save the RC procedures of the standby node.**

   Perform the same operation at the standby node as was performed at the active node.

10. **Change and register the Cmdline resource.**

    Edit the Cmdline resource sample file below and register it in the cluster system, so that the CEP engine will start automatically at a cluster switch:

    ```
    /opt/FJSVcep/HA/sample/SERVICE_BDCEP
    ```

    Edit the sample file as shown below, and then register it in the cluster system using a PRIMECLUSTER operation.

    - Change the *`Engine-Name`* part in the Cmdline resource below to the CEP engine name created at the active node and standby node, and then register it.

      ```
      STARTCMDE='/opt/FJSVcep/bin/cepstarteng -e Engine-Name'
      ```

    - To use multiple CEP engines, describe as many CEP engines as cepstarteng is to start.

    Example

    Definition example when multiple CEP engines are to be used

    ```
    (...)
    STARTCMDE='/opt/FJSVcep/bin/cepstarteng -e CepEngine1'
    STARTCMDE2='/opt/FJSVcep/bin/cepstarteng -e CepEngine2'
        (...)
        start() {
            ${STARTCMD} > /dev/null 2>&1
            ${STARTCMDE} > /dev/null 2>&1
            ${STARTCMDE2} > /dev/null 2>&1
        }
        (...)
    ```

7.2.4 Operating a Cluster Service

This section explains some points to consider in cluster service operation.

Starting and stopping

- When the active node is started, the CEP service and CEP engine will start.
- PRIMECLUSTER process monitoring will be performed for the CEP service. Therefore, do not execute cepstopserv unless it is for a cluster switch.

Sending events

- Send events to the takeover IP address.

Changing CEP engines and development assets

- Set up the CEP engine configurations and deploy the development assets (such as definition information and master data) so that the active node and the standby node have the same resource configuration.

Cluster operation and response at an active node fault

- When a hardware fault occurs at the active node and the cluster service detects the abnormality, the active node will be stopped and the CEP engine and CEP service will stop.
- After a switch to the standby node, event sending to the takeover IP address can restart.

- When an abnormality occurs at the active node, take action to resolve the hardware fault that caused the cluster switch, and then take action to allow a switch back in case an abnormality occurs at the standby node.

Chapter 8 Command Reference

This chapter explains the commands that the Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP") provides.

Command name	Function	Administrator permission
cepcollectinfo	Collects data for investigation in batch	Yes
cepconfigeng	Configures a CEP engine (*1)	Yes
cepdeployrsc	Deploys development assets, and dynamically changes rule definitions and the master data (*1)	No
cepdispeng	Displays the status of a CEP engine	No
cepdispserv	Displays the status of the CEP service	Yes
cepgetjvmopt	Displays JVM options	No
cepgetrsc	Displays development assets	No
cepsetjvmopt	Sets JVM options (*1)	Yes
cepstarteng	Starts a CEP engine (*1)	No
cepstartserv	Starts the CEP service (*2)	Yes
cepstopeng	Stops a CEP engine (*1)	No
cepstopserv	Stops the CEP service (*2)	Yes
cepundeployrsc	Undeploys a development asset (*1)	No

Yes: Must be executed by a superuser

Note

- Commands marked *1 cannot be executed simultaneously with other commands marked *1 or *2. They can be executed simultaneously with commands not marked with an asterisk.
- Commands marked *2 cannot be executed simultaneously with any other command.
- If commands are executed simultaneously (including simultaneous execution of the same command), commands executed after the first instance will end abnormally.

8.1 cepcollectinfo

Name

cepcollectinfo - Data investigation collection

Format

```
cepcollectinfo [path]
```

Function description

This command collects data for investigation for the Interstage Big Data Complex Event Processing Server, in batch.

The compressed file (tar.gz format) below will be generated in the path specified in the command.

path/collect/*yyyyMMddHHmmss*

yyyyMMddHHmmss indicates the date and time the data was collected.

Arguments

path

Specify the directory for storing the data for investigation.

If this option is omitted, the data will be stored in /tmp.

If the specified path does not exist, the command will return an error without collecting the data for investigation.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Example

Execution example

When a path is specified:

```
# cepcollectinfo /var/tmp <ENTER>
Compressing collect data is started.
Compressing collect data is finished.
Collecting is finished. (/var/tmp/collect/BDCEP_20120413141134.tar.gz)
Command cepcollectinfo executed successfully.
```

When a path is not specified (data is stored under /tmp):

```
# cepcollectinfo <ENTER>
Compressing collect data is started.
Compressing collect data is finished.
Collecting is finished. (/tmp/collect/BDCEP_20120413141134.tar.gz)
Command cepcollectinfo executed successfully.
```

8.2 cepconfigeng

Name

cepconfigeng - CEP engine configuration

Format

```
cepconfigeng -f xmlFilePath
```

Function description

This command configures a CEP engine.

It compares the content of the specified engine configuration file with that of a current CEP engine that has been configured, and then creates, changes the settings of, or deletes the CEP engine according to the differences between them.

Always save the engine configuration file for the next time the configuration content is to be changed.

When the command is executed, the prompt below is displayed - type "y" and press the Enter key to execute the change. Execution can be canceled by typing "n" or "q" and pressing the Enter key.

```
Are you sure you want to change the CEP Engine configuration? [y,n,q]:
```

Creating a CEP engine

This command creates a new CEP engine.

After the CEP engine is created, it must be started separately.

Changing CEP engine settings

This command changes the settings (Items other than CEP engine name) of a CEP engine that has been configured.

The CEP engine that is to be changed must be stopped beforehand. After this command is executed, the CEP engine must be started separately.

Deleting a CEP engine

This command deletes a CEP engine that has been configured.

The CEP engine that is to be deleted must be stopped beforehand.

If development assets have been deployed to the CEP engine that is to be deleted, the CEP engine can be deleted even if the development assets are not undeployed beforehand.

See

Refer to "8.9 cepstarteng" and "8.11 cepstopeng" for information on starting and stopping a CEP engine.

Arguments

-f *xmlFilePath*

Specify the engine configuration file that describes the definition content of the CEP engine. We recommend specifying an absolute path to avoid specification error.

See

Refer to "9.1.1 Engine Configuration File" for details.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

- A maximum of five CEP engines can be created. If six or more CEP engines are defined, this command ends abnormally.
- Specify the CEP engine name so as not to be confused with other CEP engine names. If the CEP engine in the CEP configuration file has the same name, this command will end abnormally.
- When creating a new CEP engine, always include in the engine configuration file the configured content of the CEP engine that has already been configured in order to retain that CEP engine. If this content is not included, the CEP engine that has been configured will be deleted.

Similarly, when changing CEP engine settings, always change the target settings only, and be sure to leave the other settings in the engine configuration file.

- If this command ends abnormally due to an error, it is possible that the CEP engine configuration has an inconsistency. To complete changing the CEP engine settings, remove the cause of the error and then execute the command again.
- If a CEP engine is deleted, development assets that have been deployed are undeployed automatically.
  If this command ends abnormally due to an error during execution, the status of the development asset which was undeployed prior to the error is returned to the one before execution of this command, however there may be a CEP engine configuration inconsistency. To complete the deletion of the CEP engine, remove the cause of the error and then execute the command again.
- If this command is executed while sending an event, it may end abnormally. After stopping all CEP Server event senders, execute the command again.

Example

Example of output at normal end

When CEP engines (CepEngine1 and CepEngine2) are created:

/etc/opt/FJSVcep/Engine.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<subSystemConfig xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1">
    <engineConfig id="CepEngine1">
        <logging>
            <type>bdpp</type>
            <directory>hadoop</directory>
            <loggingMaxOpenFile>6</loggingMaxOpenFile>
            <loggingRotationCycle>300</loggingRotationCycle>
        </logging>
        <socketAdapterPort>9600</socketAdapterPort>
    </engineConfig>
    <engineConfig id="CepEngine2">
        <logging>
            <type>file</type>
        </logging>
        <socketAdapterPort>9601</socketAdapterPort>
    </engineConfig>
</subSystemConfig>
```

Command execution result:

```
# cepconfigeng -f /etc/opt/FJSVcep/Engine.xml <ENTER>
Are you sure you want to change the CEP Engine configuration? [y,n,q]:y <ENTER>
Command cepconfigeng executed successfully.
```

When CepEngine1 is deleted from the CEP engines that have been configured (leaving CepEngine2):

/etc/opt/FJSVcep/Engine.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<subSystemConfig xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1">
<!--
    <engineConfig id="CepEngine1">
        <logging>
            <type>bdpp</type>
            <directory>hadoop</directory>
            <loggingMaxOpenFile>6</loggingMaxOpenFile>
            <loggingRotationCycle>300</loggingRotationCycle>
        </logging>
        <socketAdapterPort>9600</socketAdapterPort>
    </engineConfig>
```

```
-->
    <engineConfig id="CepEngine2">
        <logging>
            <type>file</type>
        </logging>
        <socketAdapterPort>9601</socketAdapterPort>
    </engineConfig>
</subSystemConfig>
```

Command execution result:

```
# cepconfigeng -f /etc/opt/FJSVcep/Engine.xml <ENTER>
Are you sure you want to change the CEP Engine configuration? [y,n,q]:y <ENTER>
Command cepconfigeng executed successfully.
```

Example of output at abnormal end

When an attempt is made to delete a CEP engine that is running.

```
# cepconfigeng -f /etc/opt/FJSVcep/Engine.xml <ENTER>
Are you sure you want to change the CEP Engine configuration? [y,n,q]:y <ENTER>
The setup processing failed. EngineId=(CepEngine) Reason=(The job is already running.)
Command cepconfigeng execution failed.
```

8.3 cepdeployrsc

Name

cepdeployrsc - Development asset deployment, and dynamic change of rule definitions and the master data

Format

```
Format 1:
cepdeployrsc resource [-o] [-e engineName] -f xmlFilePath

Format 2:
cepdeployrsc rule -h [-e engineName] -f xmlFilePath

Format 3:
cepdeployrsc master -h [-e engineName]
```

Function description

If format 1 is used, this command deploys a development asset to a CEP engine.

Specify the type of development asset (event type definition, rule definition, master definition, RDB reference definition, or SOAP listener definition) to be deployed.

If format 2 is used, this command applies the updated rule definition without stopping the CEP engine. (dynamic change of a rule definition).

If format 3 is used, this command applies the updated master data without stopping the CEP engine. (dynamic change of the master data).

When the command is executed, a prompt for confirming the deployment and dynamic change is displayed. Type "y" and press the Enter key to execute the deployment and dynamic change. Execution can be canceled by typing "n" or "q" and pressing the Enter key.

Example

Example of query (for a SOAP listener definition to be deployed):

```
Are you sure you want to deploy the SOAP listener definition?(default: y) [y,n,q]:
```

Example of query (for dynamic change of a rule definition):

```
Are you sure you want to hotdeploy the rule definition?(default: y) [y,n,q]:
```

Example of query (for dynamic change of the master data):

```
Are you sure you want to hotdeploy the master data?(default: y) [y,n,q]:
```

Arguments

resource

Specify the type of development asset. You can specify only "rule" or "master" when performing a dynamic change.

eventtype

Event type definition

rule

Rule definition

master

Master definition

rdb_ref

RDB reference definition

listener

SOAP listener definition

-o

For deployment of a development asset, this option is used to overwrite the definition content. It cannot be specified for a dynamic change.

If this option is omitted at deployment of a development asset, and if a development asset with the same ID as that specified in the definition file has already been deployed, an error message will be output and the deployment will fail.

-e *engineName*

Specify the name of the CEP engine to which the development asset is to be deployed or a dynamic change is to be made.

If only one CEP engine has been configured, this option can be omitted.

If this option is omitted, and if there are two or more configured CEP engines, an error message will be output and the deployment or dynamic change will fail.

-f *xmlFilePath*

When deploying a development asset or dynamically changing a rule definition, specify the definition file that describes the definition content of the development asset. We recommend specifying an absolute path. This option cannot be specified for dynamic change of the master data.

-h

This option dynamically changes a rule definition or the master data.

See

Refer to "9.2 Defining Development Assets" for information on the definition file.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

- A file storing detailed display results (XML format) output by cepgetrsc can also be specified as the definition file (refer to "8.7 cepgetrsc" for details).
- If format 1 is used, development asset deployment cannot be performed while the CEP engine that is the deployment target is running.
- Take the following points into account when performing dynamic change using format 2 or 3:
  - If the target CEP engine is stopped, dynamic change cannot be performed.
  - The only development assets that can be dynamically changed are rule definitions. You cannot dynamically change rule definitions that reference an undeployed event type definition, master definition, RDB reference definition, or SOAP listener definition.
  - For dynamically changing a rule definition, the rule definition with the same development asset ID as the new rule definition must have already been deployed.
  - If there is a change in the high-speed filter rule when a rule definition is dynamically changed, all master data being used on the target CEP engine is reloaded.
  - If there is a change in a complex event processing rule when a rule definition is dynamically changed, the information held in a window by the complex event processing rule is lost.
  - In the following cases, the dynamic change process is canceled and the cepdeployrsc command ends normally with a message output to the screen:
    - If a rule definition file not requiring any change is specified, the dynamic change process for the rule definition is canceled.
    - If the rule definition has not been deployed, or if a deployed rule definition does not contain a high-speed filter rule, the dynamic change process for the master data is canceled.
    - If the master definition has not been deployed, the dynamic change process for the master data is canceled.
  - While dynamic change is executing, the events to be processed are accumulated in an internal queue and processing is resumed after dynamic change succeeds.
  - If dynamic change fails due to a syntax error in a rule or any other reason, the CEP engine stops. To perform dynamic change during operations, check the operation on a different CEP engine beforehand and then perform dynamic change.

Example

Example of output at normal end

When a SOAP listener definition file (listenerdeploy.xml) is deployed to a CEP engine (CepEngine1):

```
$ cepdeployrsc listener -o -e CepEngine1 -f /tmp/listenerdeploy.xml <ENTER>
Are you sure you want to deploy the SOAP listener definition?(default: y) [y,n,q]:y <ENTER>
Command cepdeployrsc executed successfully.
```

When a rule definition file (rule.xml) is dynamically changed on a CEP engine (CepEngine1):

```
$ cepdeployrsc rule -h -e CepEngine1 -f /tmp/rule.xml <ENTER>
Are you sure you want to hotdeploy the rule definition?(default: y) [y,n,q]:y <ENTER>
Command cepdeployrsc executed successfully.
```

When the master data is dynamically changed on a CEP engine (CepEngine1):

```
$ cepdeployrsc master -h -e CepEngine1 <ENTER>
Are you sure you want to hotdeploy the master data?(default: y) [y,n,q]:y <ENTER>
Command cepdeployrsc executed successfully.
```

Example of output at abnormal end

When an attempt is made to deploy a SOAP listener definition during the execution of a command that cannot be executed at the same time:

```
$ cepdeployrsc listener -o -e CepEngine1 -f /tmp/listenerdeploy.xml <ENTER>
Are you sure you want to deploy the SOAP listener definition?(default: y) [y,n,q]:<ENTER>
Processing cannot be performed because another command is executing.
Command cepdeployrsc execution failed.
```

When the rule to be dynamically changed contains a syntax error:

```
$ cepdeployrsc rule -h -e CepEngine1 -f /tmp/rule.xml <ENTER>
Are you sure you want to hotdeploy the rule definition?(default: y) [y,n,q]:<ENTER>
The definition file contents are incorrect. Filename=(/tmp/rule.xml)
Command cepdeployrsc execution failed.
```

When an attempt is made to dynamically change a rule definition that has the ID of an undeployed development asset:

```
$ cepdeployrsc rule -h -e CepEngine1 -f /tmp/rule.xml <ENTER>
Are you sure you want to hotdeploy the rule definition?(default: y) [y,n,q]:<ENTER>
The rule definition has not been deployed. Id=(RULE01)
Command cepdeployrsc execution failed.
```

When an attempt is made to dynamically change the master data while the CEP engine is stopped:

```
$ cepdeployrsc master -h -e CepEngine1 <ENTER>
Are you sure you want to hotdeploy the master data?(default: y) [y,n,q]:<ENTER>
CEP Engine is not started. EngineId=(CepEngine1)
Command cepdeployrsc execution failed.
```

8.4 cepdispeng

Name

cepdispeng - CEP engine information display

Format

```
cepdispeng [-i] [-a | -e engineName]
```

Function description

This command displays the basic information or the operation status of a CEP engine.

Arguments

-i

This option is used to display the basic information of a CEP engine.

If this option is not specified, the operation status of the CEP engine will be displayed.

The table below explains the items displayed:

Basic information

Item name	Content	Displayed?
engineId	CEP engine name	Yes
eventtype	Development asset ID of the deployed event type definition	Conditional
rule	Development asset ID of the deployed rule definition	Conditional
master	Development asset ID of the deployed master definition	Conditional
rdb_ref	Development asset ID of the deployed RDB reference definition	Conditional
listener	Development asset ID of the deployed SOAP listener definition	Conditional

Yes: Always displayed

Conditional: Displayed if the development asset has been deployed

Point

If two or more development assets of the same definition are deployed, they are displayed in multiple lines.

Operation status

Item name	Content	Displayed?
engineId	CEP engine name.	Yes
port	Port number for socket communication (socket adapter port).	Conditional (*1)
socket	Number of simultaneous connections in socket communication.	Conditional (*1)
status_filter	Status of the CEP engine (high-speed filter). **RUN**: Running normally **STARTING**: Starting **STOP**: Stopped **STOPPING**: Stopping **ABNORMAL**: Abnormal	Yes
status_cep	Status of the CEP engine (complex event processing). **RUN**: Running normally **STARTING**: Starting **STOP**: Stopped **STOPPING**: Stopping **ABNORMAL**: Abnormal	Yes

Item name	Content	Displayed?
inEvent_filter	Number of input events for the CEP engine (high-speed filter). Indicates the number of events input to the high-speed filter. Displays a cumulative value from when the CEP engine started.	Conditional (*2)
inEvent_cep	Number of input events for the CEP engine (complex event processing). Indicates the number of events input to complex event processing. Displays a cumulative value from when the CEP engine started.	Conditional (*3)
outEvent_filter	Number of output events for the CEP engine (high-speed filter). Indicates the number of events sent to complex event processing. Displays a cumulative value from when the CEP engine started.	Conditional (*2)
outEvent_cep	Number of output events for the CEP engine (complex event processing). Indicates the number of events sent to a user-developed Web service. Displays a cumulative value from when the CEP engine started.	Conditional (*3)
logging_filter	Number of loggings for the CEP engine (high-speed filter).	Conditional (*2)
logging_cep	Number of loggings for the CEP engine (complex event processing).	Conditional (*3)
heap_filter	Java heap usage and area size for the CEP engine (high-speed filter). Displayed in the following format: Java heap usage (bytes)/area size (bytes)	Conditional (*2)
heap_cep	Java heap usage and area size for the CEP engine (complex event processing). Displayed in the following format: Java heap usage (bytes)/area size (bytes)	Conditional (*3)
useMemory_filter	Memory usage of CEP engine (high-speed filter) processes (kilobytes).	Conditional (*2)
useMemory_cep	Memory usage of CEP engine (complex event processing) processes (kilobytes).	Conditional (*3)
countRule_filter	Number of rules applied to the CEP engine (high-speed filter).	Conditional (*2)
countRule_cep	Number of rules applied to the CEP engine (complex event processing).	Conditional (*3)
countListener	Number of applied listeners. Indicates the number of SOAP listener definitions applied to the CEP engine (complex event processing).	Conditional (*3)

Yes: Always displayed

Conditional: Displayed in the following cases:

*1: If socket communication is being used (if a socket adapter is being used by the CEP engine)

*2: If the CEP engine (high-speed filter) is running normally (if status_filter is "RUN")

*3: If the CEP engine (complex event processing) is running normally (if status_cep is "RUN")

-a

This option is used to display information on all configured CEP engines.

-e *engineName*

Specify the name of the CEP engine for which information is to be displayed.

If only one CEP engine has been configured, this option can be omitted.

If this option is omitted, and if there are two or more configured CEP engines, an error message will be output and the display will fail.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Example

Example of output at normal end

When the operation statuses of all configured CEP engines are displayed:

```
$ cepdispeng -a <ENTER>
engineId         :CepEngine1
port             :9600
socket           :1
status_filter    :RUN
status_cep       :RUN
inEvent_filter   :100
inEvent_cep      :100
outEvent_filter  :100
outEvent_cep     :100
logging_filter   :100
logging_cep      :100
heap_filter      :98,725,088 / 532,545,536
heap_cep         :98,725,088 / 532,545,536
useMemory_filter :823,493,339
useMemory_cep :823,493,339
countRule_filter :1
countRule_cep    :1
countListener    :1

engineId         :CepEngine2
port             :9601
status_filter    :STOP
status_cep       :STOP
Command  cepdispeng executed successfully.
```

When the basic information of a CEP engine (CepEngine1) is displayed:

```
$ cepdispeng -i -e CepEngine1 <ENTER>
engineId         :CepEngine1
eventtype        :EVENT_01
rule             :RULE_01
master           :MASTER_01
rdb_ref          :RDBREF_01
listener         :LISTEN_01
listener         :LISTEN_02
Command cepdispeng executed successfully.
```

8.5 cepdispserv

Name

cepdispserv - CEP service status display

Format

```
cepdispserv
```

Function description

This command displays the status of the CEP service - it displays the status of each of the multiple services that make up the CEP service entity.

The services that make up the CEP service are as follows:

- Interstage Java EE DAS service
- Interstage Java EE Node Agent service
- IJServer cluster
- Interstage HTTP Server
- Apache Tomcat
- PostgreSQL

The status of each service is explained below.

Interstage Java EE DAS service

Display	Explanation
started	Running status
stopped	Stopped status
starting	Undergoing start processing
stopping	Undergoing stop processing
unknown	Unknown status

Note

"unknown" (unknown status) indicates that the status of the service could not be confirmed.

Some kind of abnormality, such as a service hang-up, is occurring, so wait for a few moments and then execute the command again to confirm the status.

Interstage Java EE Node Agent service

The display is the same as that of the Interstage Java EE DAS service.

IJServer cluster

Display	Explanation
starting	Undergoing start processing
running	Running status
stopping	Undergoing stop processing
not running	Stopped status
partially running	Degraded operation status

Note

When executing this command after cepstopserv, the message below is output, but the IJServer Cluster is stopped, so no action is required:

```
asadmin: ERROR: ISJEE_OM2997: Unable to connect to admin-server at given host:
[localhost] and port: [12001]. Please check if this server is up and running and
that the host and port provided are correct.
asadmin: ERROR: ISJEE_CLI137: Command list-clusters failed.
```

Note that in the example above, newlines have been added (lines 1 and 2) for readability only. The actual message does not have newlines.

Interstage HTTP Server

Item name	Content
Web Server Name	Web server name.
Status	Status of the Web server. **Running**: Running **Stopped**: Stopped
Configuration File	Environment configuration file.
Server Version	Server version of the Interstage HTTP Server.
Current Time	Current date and time.
Start Time	Start date and time.
Daemon Process ID	Process ID of the daemon process.
Listening Port	IP address and port number of the Web server that is to receive connection requests.

Item name	Content
	For communication using SSL protocol, "HTTPS" will be displayed for the port number. An IPv6 address will be displayed within brackets "[" and "]".
requests currently being processed	Number of requests being processed.
idle servers	Number of communication processes (threads) on standby.

Note

If "Status" is "Stopped", from "Configuration File" onwards will not be displayed.

Also, when executing this command after cepstopserv, the message below is output, but the Interstage HTTP Server is stopped, so no action is required:

```
UX:IHS: ERROR: ihs81517: The Web Server (Interstage HTTP Server) did not start. [FJapache]
```

Apache Tomcat

Display	Explanation
jsvc(pid *nnnn nnnn*) is running...	Executing. The IDs of the processes that are running will be displayed in *nnnn*.
jsvc is stopped	Stopping.

PostgreSQL

Display	Explanation
pg_ctl: server is running (PID: *nnnn*) /..../bin/postgres -D /..../data	Executing. The ID of the process that is running will be displayed in *nnnn*.
pg_ctl: no server running	Stopping.

Arguments

None

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Example

Example of output at normal end

When all services are running normally:

```
# cepdispserv <ENTER>
Interstage Java EE DAS service status.
ijdasstat
Name                             Status
-----------------------------------------
Interstage Java EE DAS           started

Interstage Java EE Node Agent service status.
ijnastat
Name                             Status
-----------------------------------------
Interstage Java EE Node Agent    started

IJServer Cluster status.
CEPAgentIJServerCluster running
CepEngine_flt not running
CepEngine_cep not running
Command list-clusters executed successfully.

Interstage HTTP Server status.
ihsdisp

Web Server Name   : FJapache
Status            : Running
Configuration File: /opt/FJSVihs/servers/FJapache/conf/httpd.conf
Server Version    : FJapache/10.0
Current Time      : Monday, 09-Jul-2012 01:25:17
Start Time        : Sunday, 08-Jul-2012 15:20:31
Daemon Process ID : 2267
Child Process ID  : 2272 2273 2274 2275 2276
Listening Port    : [::]:80
0 requests currently being processed, 5 idle servers


Apache Tomcat status.
/sbin/service FJSVcep-rest status
jsvc (pid 2099 2096) is running...

PostgreSQL status.
su - bdcep_postgres -c "/opt/FJSVcep/postgres/packages/FJSVpgs83/bin/pg_ctl -D /var/opt/FJSVcep/
postgres/data status"
pg_ctl: server is running (PID: 2088)
/opt/FJSVcep/postgres/packages/FJSVpgs83/bin/postgres "-D" "/var/opt/FJSVcep/postgres/data"

Command cepdispserv executed successfully.
```

8.6 cepgetjvmopt

Name

cepgetjvmopt - JVM options display

Format

```
cepgetjvmopt function -e engineName
```

Function description

This command displays the JVM options of a CEP engine.

The JVM options that are displayed are shown below.

Item	Content
xmxSize	Maximum value of memory allocation pool
xmsSize	Initial value of memory allocation pool
maxPermSize	Maximum value of permanent generation area

Arguments

function

Specify the feature for which the JVM options are to be displayed.

filter

High-speed filter

cep

Complex event processing

-e *engineName*

Specify the name of the CEP engine for which the JVM options are to be displayed.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Example

Example of output at normal end

When the JVM options of the high-speed filter of a CEP engine (CepEngine1) are displayed:

```
$ cepgetjvmopt filter -e CepEngine1 <ENTER>
xmxSize               :5120m
xmsSize               :256m
maxPermSize           :96m
Command cepgetjvmopt executed successfully.
```

8.7 cepgetrsc

Name

cepgetrsc - Development assets display

Format

```
cepgetrsc resource [-e engineName] [-n resourceId]
```

Function description

This command displays a list of or details of the development assets that have been deployed.

Arguments

resource

Specify the type of development asset.

eventtype

Event type definition

rule

Rule definition

master

Master definition

rdb_ref

RDB reference definition

listener

SOAP listener definition

-e *engineName*

Specify the name of the CEP engine for which the development assets are to be displayed.

If only one CEP engine has been configured, this option can be omitted.

If this option is omitted, and if there are two or more configured CEP engines, an error message will be output and the display will fail.

-n *resourceId*

Specify the development asset ID to be displayed in detail.

When this option is specified, the definition content of the development asset is displayed in XML format.

See

Refer to "9.2 Defining Development Assets" for information on display results in XML format.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

- If diverting the results of the detailed display (XML format), modify each item accordingly.
- The results of this command include output messages in addition to XML format data.
  If diverting the command results by redirecting them to a file, delete unnecessary messages before using the results.

Example

Example of output at normal end

When the SOAP listener definitions that have been deployed to a CEP engine (CepEngine1) are displayed as a list:

```
$ cepgetrsc listener -e CepEngine1 <ENTER>
LISTEN_01
LISTEN_02
Command cepgetrsc executed successfully.
```

When the event type definitions that have been deployed to a CEP engine (CepEngine1) are displayed in detail:

```
$ cepgetrsc eventtype -e CepEngine1 -n EVENT01 <ENTER>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventType xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="EVENT01">
    <comment>Event type definition</comment>
    <type>CSV</type>
    <xmlSchema></xmlSchema>
    <csvColumn>
        <column name="memberID" type="string"/>
        <column name="areaID" type="string"/>
        <column name="status" type="string"/>
    </csvColumn>
    <root></root>
    <useLogging>false</useLogging>
    <loggingTableName></loggingTableName>
    <useCep>true</useCep>
</eventType>

Command cepgetrsc executed successfully.
```

Example of output at abnormal end

When an attempt is made to display an event type definition that has not been deployed to a CEP engine (CepEngine2):

```
$ cepgetrsc eventtype -e CepEngine2 -n EVENT02 <ENTER>
The event type definition does not exist. Id=(EVENT02)
Command cepgetrsc execution failed.
```

The following information will be added to the error message in this example:

Id

Development asset ID that was specified

8.8 cepsetjvmopt

Name

cepsetjvmopt - JVM options set

Format

```
cepsetjvmopt function [-xmx xmxSize] [-xms xmsSize] [-xxmp permSize] -e engineName
```

Function description

This command sets the JVM options of a CEP engine.

Arguments

function

Specify the feature for which the JVM options are to be set.

filter

High-speed filter

cep

Complex event processing

-xmx *xmxSize*

This option is used to set the maximum value of the memory allocation pool.

If this option is omitted, the default (2048m) will be used.

If the value specified here is less than the value specified in the -xms option, an error message will be output when the CEP engine is started, and the start will fail.

The following characters can be specified as units:

To specify KB (kilobytes): "**k**" or "**K**"

To specify MB (megabytes): "**m**" or "**M**"

If the unit is omitted, the specification will be in bytes. Specify a value greater than 1 MB that is a multiple of 1024.

-xms *xmsSize*

This option is used to set the initial value of the memory allocation pool.

If this option is omitted, the default (512m) will be used.

If the value specified here is less than 2624 KB, an error message will be output when the CEP engine is started, and the start will fail.

The following characters can be specified as units:

To specify KB (kilobytes): "**k**" or "**K**"

To specify MB (megabytes): "**m**" or "**M**"

If the unit is omitted, the specification will be in bytes. Specify a value greater than 1 MB that is a multiple of 1024.

-xxmp *permSize*

This option is used to set the maximum value of the permanent generation area.

If this option is omitted, the default (192m) will be used.

If the value specified here is less than 20.75 MB, then 20.75 MB will be used.

The following characters can be specified as units:

To specify KB (kilobytes): "**k**" or "**K**"

To specify MB (megabytes): "**m**" or "**M**"

If the unit is omitted, the specification will be in bytes. Specify a value greater than 1 MB that is a multiple of 1024.

-e *engineName*

Specify the name of the CEP engine for which the JVM options are to be set.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

This command may be used for the running CEP engine, however, to apply settings, the target CEP engine must be restarted.

Example

Example of output at normal end

When the JVM options of the high-speed filter of a CEP engine (CepEngine1) are set:

```
# cepsetjvmopt filter -xmx 5120m -xms 256m -xxmp 96m -e CepEngine1 <ENTER>
Command cepsetjvmopt executed successfully.
```

8.9 cepstarteng

Name

cepstarteng - CEP engine start

Format

```
cepstarteng [-e engineName]
```

Function description

This command starts a CEP engine.

Arguments

-e *engineName*

Specify the name of the CEP engine to be started.

If only one CEP engine has been configured, this option can be omitted.

If this option is omitted, and if there are two or more configured CEP engines, an error message will be output and the start will fail.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

- Before this command is executed, cepdeployrsc must be used to deploy at least one event type definition. If an event type definition has not been deployed, an error message will be output when the CEP engine is started, and the start will fail.
- If RDB collaboration or Terracotta collaboration is implemented, the collaboration destination server must have been started before this command is executed. If the collaboration destination server is stopped, an error message will be output when the CEP engine is started, and the start will fail.
- This command monitors the start of the CEP engine until it has completed, and the command ends normally if it can confirm that the start has completed.
- If this command is not completed within 180 seconds, force the CEP engine to stop. If the forced stop is then completed within 180 seconds, a message is output stating that the startup processing has timed out and that this command has ended abnormally. If the forced stop is not completed within 180 seconds, a message is output stating that the forced stop has failed and that this command has ended abnormally.

Example

Example of output at normal end

When a CEP engine (CepEngine1) is started:

```
$ cepstarteng -e CepEngine1 <ENTER>
Command cepstarteng executed successfully.
```

Example of output at abnormal end

When an attempt is made to start a CEP engine that does not exist:

```
$ cepstarteng -e CepEngine2 <ENTER>
An incorrect value was entered. Reason=(The engine does not exist.)
Command cepstarteng execution failed.
```

The following information will be added to the error message in this example:

Reason

Cause of the error (reason for the failure to start the CEP engine)

8.10 cepstartserv

Name

cepstartserv - CEP service start

Format

```
cepstartserv
```

Function description

This command starts the CEP service - it starts in batch the multiple services that make up the CEP service entity.

Start processing for services that are already started will be skipped.

The services started by this command are as follows:

- Interstage Java EE DAS service
- Interstage Java EE Node Agent service
- Interstage Management Console
- Interstage HTTP Server
- PostgreSQL
- Apache Tomcat

Arguments

None

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

- If the start of even one of the services that make up the CEP service fails, an error message will be output and the start of the CEP service will fail.
  To complete the start of the CEP service, remove the cause of the error and then execute the command again.
- The CEP engine is not started by this command.
- BDCEP uses cron for CEP engine resource usage acquisition. This command does not start cron, it is normally started automatically when OS starts (refer to "6.3.3 Applying Updates" for information on how to start cron manually).

Example

Example of output at normal end

When all services start successfully:

```
# cepstartserv <ENTER>
Starting Interstage Java EE DAS service.
ijdasstart: INFO: ijdas10000: Interstage Java EE DAS service has started.
Name                             Status
-----------------------------------------
Interstage Java EE DAS           started

Starting  Interstage Java EE Node Agent service.
ijnastart: INFO: ijna10000: Interstage Java EE Node Agent service has started.
Name                             Status
-----------------------------------------
Interstage Java EE Node Agent    started

Starting Interstage Management Console.
UX:ismngconsolestart: INFO: is40041: The service has been activated normally.

Starting Interstage HTTP Server.
UX:IHS: INFO: ihs01000: The command terminated normally.

Starting PostgreSQL.
su - bdcep_postgres -c "/opt/FJSVcep/postgres/packages/FJSVpgs83/bin/pg_ctl -D /var/opt/FJSVcep/
postgres/data -w start"
waiting for server
to start...LOG:  database system was shut down at 2011-11-21 15:03:21 JST
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
 done
server started

Starting Apache Tomcat.
/etc/init.d/FJSVcep-rest start
Starting Tomcat:                                           [  OK  ]

Command cepstartserv executed successfully.
```

Example of output at abnormal end

When the Interstage Java EE DAS service cannot be started:

```
# cepstartserv <ENTER>
Starting Interstage Java EE DAS service.
```

```
ERROR: ijdas10002: Interstage Java EE DAS service cannot be started.
Command cepstartserv execution failed.
```

8.11 cepstopeng

Name

cepstopeng - CEP engine stop

Format

```
cepstopeng [-e engineName]
```

Function description

This command stops a CEP engine.

Arguments

-e *engineName*

Specify the name of the CEP engine to be stopped.

If only one CEP engine has been configured, this option can be omitted.

If this option is omitted, and if there are two or more configured CEP engines, an error message will be output and the stop will fail.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

This command monitors the stop of the CEP engine until it has completed, and the command ends normally if it can confirm that the stop has completed.
If the stop has not completed within a predetermined time (within 180 seconds), an error message will be output and the stop will fail with a timeout.

Example

Example of output at normal end

When a CEP engine (CepEngine1) is stopped:

```
$ cepstopeng -e CepEngine1 <ENTER>
Command cepstopeng executed successfully.
```

Example of output at abnormal end

When stop processing has not completed within a predetermined time (when a timeout has occurred):

```
$ cepstopeng -e CepEngine2 <ENTER>
ERROR: cep30203e: Failed to stop Collection Engine. EngineId=(CepEngine2) JobId=(CepEngine2
-cepjobnet-20120710130118-CepEngine2) Reason=(A timeout occurred.(localhost_FRT))
Command cepstopeng execution failed.
```

Note that in the example above, a newline has been added (line 2) for readability only. The actual message does not have newlines.

The following information will be added to the error message in this example:

EngineId

Name of the CEP engine to be stopped

JobId

The Job ID, in the following format:

```
cepEngineName-cepjobnet-cepEngineStartDatetime(yyyyMMddHHmmss)-cepEngineName
```

Reason

Error details (reason for the failure to stop the CEP engine)

8.12 cepstopserv

Name

cepstopserv - CEP service stop

Format

```
cepstopserv
```

Function description

This command stops the CEP service - it stops in batch the multiple services that make up the CEP service entity.

Stop processing for those services that are already stopped will be skipped.

This command also stops a CEP engine that has been started.

The services stopped by this command are as follows:

- CEP engine
- Apache Tomcat
- PostgreSQL
- Interstage HTTP Server
- Interstage Management Console
- Interstage Java EE Node Agent service
- Interstage Java EE DAS service
- Interstage

Arguments

None

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Point

Executing this command will stop all CEP engines that are running.

Note

- If the stop of one or more of the services fails, an error message will be output and the stop of the CEP service will fail.
  To complete the stop of the CEP service, remove the cause of the error and then execute the command again.
- BDCEP uses cron for CEP engine resource usage acquisition. This command does not stop cron (refer to "6.3.3 Applying Updates" for information on how to stop cron).
- If this command is executed twice in a row, the message below is output, but the CEP engine or Apache Tomcat is stopped, so no action is required:

```
Stopping Engines.
asadmin: ERROR: ISJEE_OM2997: Unable to connect to admin-server at given host:
[localhost] and port: [12001]. Please check if this server is up and running and
that the host and port provided are correct.
asadmin: ERROR: ISJEE_CLI137: Command list-clusters failed.

Stopping Apache Tomcat.
/etc/init.d/FJSVcep-rest stop
Shutting down Tomcat:                                      [Fail]
```

Note that in the example above, newlines have been added (lines 2 and 3) for readability only. The actual message does not have newlines.

Example

Example of output at normal end

When all services stop successfully:

```
# cepstopserv <ENTER>
Stopping Engines.
The engine is not started. EngineId=(CepEngine)
Command cepstopeng executed successfully.

Stopping Apache Tomcat.
/etc/init.d/FJSVcep-rest stop
Shutting down Tomcat:                                      [  OK  ]

Stopping PostgreSQL.
```

```
su - bdcep_postgres -c "/opt/FJSVcep/postgres/packages/FJSVpgs83/bin/pg_ctl -D /var/opt/FJSVcep/
postgres/data stop"
waiting for server to shut down.... done
server stopped

Stopping Interstage HTTP Server.
UX:IHS: INFO: ihs01000: The command terminated normally.

Stopping Interstage Management Console.
UX:ismngconsolestop: INFO: is40042: The service has been terminated normally.

Stopping Interstage Java EE Node Agent service.
ijnastop: INFO: ijna10001: Interstage Java EE Node Agent service has stopped.
Name                             Status
-----------------------------------------
Interstage Java EE Node Agent    stopped

Stopping Interstage Java EE DAS service.
ijdasstop: INFO: ijdas10001: Interstage Java EE DAS service has stopped.
Name                             Status
-----------------------------------------
Interstage Java EE DAS           stopped

Stopping Interstage.
UX:isstop: INFO: is30160:INTERSTAGE terminated normally.

Command cepstopserv executed successfully.
```

Example of output at abnormal end

When the Interstage Java EE DAS service cannot be stopped.

```
# cepstopserv <ENTER>
(...)
Stopping Interstage Java EE DAS service.
ERROR: ijdas10003: Interstage Java EE DAS service cannot be stopped.
Command cepstopserv execution failed.
```

8.13 cepundeployrsc

Name

cepundeployrsc - Development asset undeployment

Format

```
cepundeployrsc resource [-e engineName] -n resourceId
```

Function description

This command undeploys a development asset that has been deployed.

Specify the type of development asset (event type definition, rule definition, master definition, RDB reference definition, or SOAP listener definition) to be undeployed.

When the command is executed, a prompt is displayed - type "y" and press the Enter key, or simply press the Enter key, to execute the undeployment. Execution can be canceled by typing "n" or "q" and pressing the Enter key.

Example

Example of query (for an event type definition):

```
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:
```

Arguments

resource

Specify the type of development asset.

eventtype

Event type definition

rule

Rule definition

master

Master definition

rdb_ref

RDB reference definition

listener

SOAP listener definition

-e *engineName*

Specify the name of the CEP engine to which the development asset to be undeployed has been deployed.

If only one CEP engine has been configured, this option can be omitted.

If this option is omitted, and if there are two or more configured CEP engines, an error message will be output and the undeployment will fail.

-n *resourceId*

Specify the development asset ID to be undeployed.

End status

The following status codes are returned:

0

Normal end

8

Abnormal end

Note

If the CEP engine that is the undeployment target is running, undeployment cannot be performed.

Example

Example of output at normal end

When an event type definition that has been deployed to a CEP engine (CepEngine1) is undeployed:

```
$ cepundeployrsc eventtype -e CepEngine1 -n EVENT_01 <ENTER>
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:y <ENTER>
Command cepundeployrsc executed successfully.
```

Example of output at abnormal end

When an attempt is made to undeploy an event type definition that has been deployed to a CEP engine (CepEngine2) that is running:

```
$ cepundeployrsc eventtype -e CepEngine2 -n EVENT_02 <ENTER>
Are you sure you want to undeploy the event type definition?(default: y) [y,n,q]:y <ENTER>
Because the definition is used, it cannot be deleted.  Definition=(instream) Id=(EVENT_02-CepEngine2)
Command cepundeployrsc execution failed.
```

The following information will be added to the error message in this example:

Id

Development asset ID and target CEP engine name to be undeployed.

Chapter 9 Definition File Reference

This chapter explains the definition files that the Interstage Big Data Complex Event Processing Server (hereafter referred to as "BDCEP") uses.

9.1 Defining a CEP Engine

This section explains the definition file for configuring a CEP engine (engine configuration file).

An engine configuration file for use by the initial CEP engine (/etc/opt/FJSVcep/cep/sample_eng.xml) is created automatically at installation. To change the configuration details, make a copy of this file and then edit the copy.

Creating, deleting or changing the setting of CEP engine is done by executing cepconfigeng with specified engine configuration file.

Point

- The engine configuration file is a text file in XML format.
- The items to be set are specified as XML element and attribute values.
- The character encoding for the engine configuration file is UTF-8.

Note

Creating, deleting, or changing the settings of a CEP engine can also be done by editing the engine configuration file of the initial CEP engine, but to prevent configuration details being lost due to an editing mistake, do not use this method.

9.1.1 Engine Configuration File

This section explains the settings of the engine configuration file.

The engine configuration file is an XML file that has "subSystemConfig" as the root element. The configuration information of one CEP engine is described in an "engineConfig" element. When multiple CEP engines are being created, the configurations of all the CEP engines must be described in one engine configuration file (using consecutive "engineConfig" elements).

Also, if Logging is to be used in the input adapter, describe the "logging" subelement in the "engineConfig" element.

The items to be set in the engine configuration file are shown below:

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of the "engineConfig" element)	CEP engine name	ID that uniquely identifies the CEP engine.	Up to 20 alphanumeric characters, including underscores (_). Note: The first character must be a letter.	Mandatory
type (*1)	Logging type	Log destination. **bdpp**: Output to a Hadoop system. **file**: Output to the engine log.	See Explanation.	(*2)
directory (*1)	Directory name	Directory of the Hadoop system that is the log destination. Specify one of the following:	Up to 1023 alphanumeric	(*3)

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
		- Name of the directory (not its full path). - "/".	characters, including forward slashes (/). Note: Use only alphanumeric characters for subdirectory names.	
loggingMaxOpenFile (*1)	Number of open log files	Number of files to be simultaneously opened for the Hadoop system. The default is 6.	1 to 122.	(*4)
loggingRotationCycle (*1)	Logging cycle time	Time from when the event log files are opened until those files are renamed as files that can be analyzed (using the ".done" extension). The default is 300 (seconds).	1 to 2592000. (unit: seconds)	(*4)
socketAdapterPort	Socket adapter port	Reception port number to be used by the socket adapter.	1 to 65535.	(*5)

*1: Subelement of the logging element

*2: Mandatory if logging is to be used

*3: Mandatory if logging is to be used and the logging type is "bdpp"

*4: Optional if logging is to be used and the logging type is "bdpp"

*5: Specified if socket communication is to be used by the input adapter

Point

- The log destination (path) in a Hadoop system is specified in the event type definition. Therefore, if "bdpp" is specified in the logging type, there is no need to specify the log destination (path).
- If "bdpp" is to be specified in the log destination, the Interstage Big Data Parallel Processing Server (hereafter, referred to as "BDPP") must be set up beforehand (refer to "4.4.2 Setup of Hadoop Collaboration" for details).

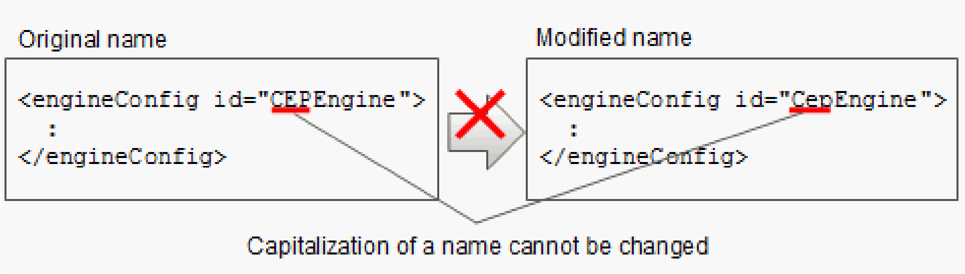
Note

- Up to five CEP engines ("engineConfig" elements) can be described in the engine configuration file.
- Specify a CEP engine name that is different from other CEP engine names described in the engine configuration file. CEP engine names that differ only in the capitalization of letters are treated as duplicate names. The following example specifies duplicate names:

```
<engineConfig id="CEPEngine">
  :
</engineConfig>
<engineConfig id="CepEngine">
  :
</engineConfig>
```

- Do not specify a name that is identical to an existing CEP engine name, except for the capitalization of letters.

  The following example specifies the name "CepEngine", which differs from the existing CEP engine name "CEPEngine" only in terms of capitalization:


Original name
<engineConfig id="CEPEngine">
:
</engineConfig>
Modified name
<engineConfig id="CepEngine">
:
</engineConfig>
Capitalization of a name cannot be changed

- Specify a port number that is not being used by the system as the socket adapter port.
- The same port number can be specified for the socket adapter port of multiple CEP engines, but the CEP engines cannot be started simultaneously.

Example

When two CEP engines (CepEngine1 and CepEngine2) are configured

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<subSystemConfig xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1">
    <engineConfig id="CepEngine1">
        <logging>
            <type>bdpp</type>
            <directory>hadoop</directory>
            <loggingMaxOpenFile>6</loggingMaxOpenFile>
            <loggingRotationCycle>300</loggingRotationCycle>
        </logging>
        <socketAdapterPort>9600</socketAdapterPort>
    </engineConfig>
    <engineConfig id="CepEngine2">
        <logging>
            <type>file</type>
        </logging>
        <socketAdapterPort>9601</socketAdapterPort>
    </engineConfig>
</subSystemConfig>
```

In this example, the following settings are used to configure the CEP engines:

CepEngine1

Uses logging and records events in a Hadoop system (Directory name: hadoop). The number of logging files that can be opened is 6, and the logging cycle time is 300 seconds.

Uses the socket adapter listening at port 9600.

CepEngine2

Uses logging and records events in the engine log.

Uses the socket adapter listening at port 9601.

9.2 Defining Development Assets

This section explains the definition files (development assets) for using the various CEP service features.

There are four kinds of development assets, as shown below, and each has an identifier known as a development asset ID, which must be unique per definition type.

cepdeployrsc deploys a development asset to a CEP engine, and cepundeployrsc undeploys a development asset.

Definition type	Definition overview	Used by feature				Mandatory/ Optional
		Input adapter	High-speed filter	Complex event processing	Output adapter	
Event type definition	Defines the data structure of the input event data. Also sets Logging in the input adapter.	Used	Used	Used		(*1)
Rule definition	Describes high-speed filter rules and complex event processing rules.		Used	Used	Used	(*2)
Master definition	Sets the master data used by the High-speed Filter.		Used			(*3)
RDB reference definition	Sets the connection destination information about the database to be referenced using RDB Collaboration.			Used		(*4)
SOAP listener definition	Sets the send destination information when SOAP communication is to be used to notify user-developed Web service events.				Used	(*5)

*1: At least one development asset must be deployed.

*2: Deploy if the high-speed filter or complex event processing is to be used.

*3: Deploy if the master data is to be referenced in the high-speed filter.

*4: Deploy if RDB collaboration is to be used in Complex Event Processing.

*5: Deploy if the SOAP listener is to be used in the output adapter.

Point

- Each definition file is a text file in XML format (except for the master data).
- The items to be set are specified as XML element and attribute values.
- The character encoding for each definition file is UTF-8.

9.2.1 Event Type Definition File

This section explains the setup file for describing the event type definition (event type definition file).

The event type definition file is an XML file that has "eventType" as the root element. An event type definition file is created for each event type. The items to be described will vary according to the format of the event data (CSV or XML format).

The items to be set in the event type definition file are shown below:

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of "eventType")	Development asset ID	ID unique in the deployment CEP engine. Used as the event type name specified in high-speed filter rules and as the event stream name specified in complex event processing rules.	Up to 39 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
comment	Comment	Explanation of this definition.	Up to 1,000 characters.	Optional
type	Format	Format of the event data. **XML**: Events are in XML format. **CSV**: Events are in CSV format.	See Explanation.	Mandatory
xmlSchema	XML schema	XML schema that represents the structure of the event data. This is described in XML schema language. Therefore, escape must be performed in a way that cannot be interpreted as a markup specification. For simple description, describe using a CDATA section instead of escaping.	Up to 1,048,576 characters.	(*1)
root	Root element	Name of the root element of the event. The root element is one of the elements defined in the XML schema. Specify unique root element names for all event type definitions.	Up to 512 characters.	(*1)
csvColumn	CSV column information	Column information that represents the structure of the event data. Describe at least one "column" element (as described below).	Follows the column element (as described below). There is no limit to the number of column elements.	(*2)
useLogging	Whether to use logging	Whether to use logging for pre-processing events of the high-speed filter received from the input adapter. **true**: Use. **false**: Do not use.	See Explanation.	Mandatory
loggingTableName	Log storage area	Absolute path of the log storage area for storing events. This is used to store events received by the input adapter. Even if logging an event in an engine log (when the engine structure file type element is set as file), for event identification set the virtual path	Up to 255 characters.	(*3)

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
		name (for example, "/ *eventTypeID*").		
useCep	Whether to use complex event processing	Controls whether to use complex event processing. **true**: Use. **false**: Do not use.	See Explanation.	Mandatory

*1: Mandatory if "XML" is specified as the format

*2: Mandatory if "CSV" is specified as the format

*3: Mandatory if logging is to be used

If "CSV" is specified as the format, describe the CSV column information shown below in the "csvColumn" element:

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
column	column	Describe as many of these as the number of event (CSV format) columns. The column elements must be specified in the same order as in the event data.	(empty element)	Mandatory
name (attribute of "column")	Item name	Item name of the CSV column. Used as the item name specified in high-speed filter rules and as the property name specified in complex event processing rules.	Refer to "9.6 Characters Allowed in Item, Tag and Attribute Names".	Mandatory
type (attribute of "column")	Item type	Data type of the CSV column. **string**: String. **boolean**: Boolean value (true/false). **byte**: 8-bit signed integer. **int**: 32-bit signed integer. **long**: 64-bit signed integer. **float**: 32-bit float. **double**: 64-bit double precision float.	See Explanation.	Mandatory

The data types of CSV columns are converted to the appropriate type for complex event processing rules, as shown below:

Data type of CSV column	Data type in complex event processing rules
string	string
boolean	bool/boolean
byte	byte
int	int/integer
long	long
float	float

Data type of CSV column	Data type in complex event processing rules
double	double

Point

If the high-speed filter is being used, the event types of the events passed to complex event processing and of the input events will vary, unless only extraction (filtering) is being used. In this case, a separate event type definition for complex event processing must be deployed.

Note

- Up to 32 event type definitions can be deployed to one CEP engine.
- You can choose whether to include XML declarations in the XML schema. If you include them, do not insert newlines or whitespace characters between the xmlSchema start tag and the XML declaration.
- You can define the order of elements by using the "sequence" element in the XML schema, but no error occurs even if the XML elements in input events do not conform to the order specified in the "sequence" element.
- Regardless of the CEP engine to deploy event type definitions, set a unique root element name for all event type definitions.
- Regarding the useCep attribute (which determines whether to use complex event processing):

  Set "false" if only logging is to be used by the input adapter. An investigation of the content of input event data is considered prior to starting operation.
- The column elements must be specified in the same order as in the event data.
- When using cepgetrsc to display details of the development asset, the displayed XML schema uses the same format as used by the CDATA section.

Example

When XML format is selected

Target event data (XML):

```
<?xml version="1.0" encoding="UTF-8"?><messagedata xmlns="http://dataaccesscontrol.sspf.
fujitsu.com/namespace/xmlmessage"><memberID>MEM0001</memberID><areaID>1010</areaID><sta
tus>1</status></messagedata>
```

Note that in the example above, newlines have been added for readability only. The actual data does not have newlines.

Event type definitions:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventType xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="EVENTTYPE_01">
    <comment>Event type definition_01</comment>
    <type>XML</type>
    <xmlSchema>
        <![CDATA[
            <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://dataaccesscontrol.sspf.fujitsu.com/namespace/xmlmessage"
            targetNamespace="http://dataaccesscontrol.sspf.fujitsu.com/namespace/xmlmessage">
            <xs:element name="messagedata">
            <xs:complexType>
            <xs:sequence>
            <xs:element name="memberID" type="xs:string" />
            <xs:element name="areaID" type="xs:string" />
            <xs:element name="status" type="xs:string" />
            </xs:sequence>
            </xs:complexType>
            </xs:element>
            </xs:schema>
        ]]>
    </xmlSchema>
    <root>messagedata</root>
    <useLogging>true</useLogging>
    <loggingTableName>/echonet</loggingTableName>
    <useCep>true</useCep>
</eventType>
```

When CSV format is selected

Target event data (CSV):

```
"MEM0001","1010","1"
```

Event type definitions:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventType xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="EVENTTYPE_02">
    <comment>Event type definition_02</comment>
    <type>CSV</type>
    <xmlSchema></xmlSchema>
    <csvColumn>
        <column name="memberID" type="string" />
        <column name="areaID" type="string" />
        <column name="status" type="string" />
    </csvColumn>
    <useLogging>false</useLogging>
    <useCep>true</useCep>
</eventType>
```

In this example, the event types below are defined.

EVENTTYPE_01

Using the input adapter, the event data is output to a log in "`/echonet`".

This event data is passed to complex event processing (only if it is not being filtered by the high-speed filter).

EVENTTYPE_02

In the input adapter, the event data is not output to a log.

This event data is passed to complex event processing (only if it is not being filtered by the high-speed filter).

9.2.2 Rule Definition File

This section explains the setup file for describing a rule definition (rule definition file).

The rule definition file is an XML file that has "rule" as the root element. The rule definition file describes the rules for the high-speed filter and complex event processing.

The items to be set in the rule definition file are shown below:

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of "rule")	Development asset ID	ID unique in the deployment CEP engine.	Up to 39 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
comment	Comment	Explanation of this definition.	Up to 1,000 characters.	Optional
filter	Filter rule	Description of a high-speed filter rule. Use a method such as a CDATA section if any part of the description can be interpreted as a markup specification.	Up to 1,048,576 characters.	(*1)
statements	Complex event processing rule	Description of a complex event processing rule. Use a method such as a CDATA section if any part of the description can be interpreted as a markup specification.	Up to 1,048,576 characters.	(*2)

*1: Mandatory if a high-speed filter rule is to be described.

*2: Mandatory if a complex event processing rule is to be described.

Point

- Filter rule options (such as "`SkipChar`", which specifies strings to be excluded as search targets), must be described at the beginning of the filter rule (refer to Chapter 2, "Filter Rule Language Reference" in the *Developer's Reference* for information on filter rule options).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rule xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RULE_02">
    <comment>Rule definition_02</comment>
    <filter>
        <![CDATA[
            @SkipChar("\n")
            @SeparateChar("\t")
            @ANKmix(true)
            @KNJmix(true)

            on EVENTTYPE_02 {
                (...)
            }
        ]]>
    </filter>
</rule>
```

- A huge number of rules can be described in a single rule definition (as long as the number of characters is not exceeded), but when rules become complex, this can negatively affect their ability to be maintained and referenced. When creating rule definitions, consider creating separate rule definitions for the high-speed filter and for complex event processing, and dividing rule definitions into smaller ones that are meaningful and cohesive.

Note

- Up to 32 rule definitions can be deployed to one CEP engine.
- When deploying multiple rule definitions to one CEP engine, specify one complex event processing rule where there is a dependency relationship (for example, a SELECT statement that specifies an event stream using an INSERT INTO clause, and a SELECT statement containing a FROM clause that specifies the event stream) in one rule definition.
- When using cepgetrsc to display details of the development asset, the displayed filter rules and complex event processing rules use the same format as used by the CDATA section.

See

- Refer to Chapter 2, "Filter Rule Language Reference" in the *Developer's Reference* for information on how to describe filter rules.
- Refer to Chapter 1, "Complex Event Processing Language Reference" in the *Developer's Reference* for information on how to describe complex event processing rules.

Example

Definition example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rule xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RULE_01">
    <comment>Rule definition_01</comment>
    <filter>
        <![CDATA[
            on EVENTTYPE_01 {
              if ($status == 'Walking') then output() as EVENTTYPE_01;
            }
        ]]>
    </filter>
    <statements>
        <![CDATA[
            @SoapListener('LISTEN_01')
            @DebugLogListener
            select * from EVENTTYPE_01 where areaID = '1010';
        ]]>
    </statements>
</rule>
```

In this example, the rules shown below are described.

High-speed filter rule

If the status item contents (string) extracted from the "`EVENTTYPE_01`" event type input event is "`Walking`", it is transferred to Complex Event Processing.

Complex event processing rule

This rule notifies the SOAP listener of event data with the "`EVENTTYPE_01`" event type, and simultaneously outputs debug information to the engine log.

9.2.3 Master Definition File

This section explains the setup file for describing the master definition (master definition file).

The master definition file is an XML file that has "master" as the root element. A master definition file is created for each unit of master data. The master data consists of a schema file and at least one data file, and is matched to a master definition file.

The master data is read when the CEP engine to which the master definition is deployed starts.

The items to be set in the master definition file are shown below:

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of "master")	Development asset ID	ID unique in the deployment CEP engine.	Up to 39 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
comment	Comment	Explanation of this definition.	Up to 1,000 characters.	Optional
schemaFile	Schema file	Full name the schema file.	Up to 1,023 bytes.	Mandatory
dataFile	Data file	Full name of the data file. If there are multiple data files, specify multiple "dataFile" elements. Alternatively, using an absolute path, specify the directory where the data file is located. In this case, all files in the directory will be read as data files.	Up to 1,023 bytes. Note: If specifying a directory, set this so that the directory name plus the file name does not exceed 1,023 bytes.	Mandatory
skipHeader	Skip specification	Whether to skip the first line in the data file when the item name is described in the first line of the data file. **true**: Assume the first line is not data (ignore). **false**: Assume the first line is data (default).	See Explanation.	Optional

The schema file and data file are text files in CSV format (refer to "9.7 CSV Format Supported" for details). The character encoding for the files is UTF-8.

Schema file

In a schema definition file, one or more item names are described using one record.

Refer to "9.6 Characters Allowed in Item, Tag and Attribute Names" for information on the characters that can be specified in item names.

Data file

In a data file, one line corresponds to one entry of master data.

In each line, describe values (data) that correspond to the schema file items.

If the number of data items is greater than the number of schema information items, the data will be regarded invalid.

Note

- Up to 32 master definitions can be deployed to one CEP engine.
- Up to 100 "dataFile" elements can be specified in one master definition. If this limit is exceeded, locate the data files in any directory and then specify the path of the directory.
- If the number of data file items is less than the number of schema file items, empty strings ("") will be used in the missing items. If the number of data file items is greater, the CEP engine will fail to start.
- If data files are specified using a directory path, ordinary files and symbolic links (with link destinations of ordinary files) located in the directory are read as data files. Aside from subdirectories, do not locate files other than ordinary files and symbolic links (such as named pipes and special files) in the directory.
- The master data is opened in memory when the CEP engine starts. Therefore, memory is consumed in proportion to the master data size.
- After updating the master data, you must restart (stop and then start) the CEP engine or dynamically change the master data using the cepdeployrsc command in order to reflect the update.
- Read permissions for the engine execution user must be given in the schema file and data files.

Example

- Example of a schema file

  /var/tmp/SchemaFile01.csv

```
"Kbn","Number","Code","Name","Value","Total","Biko"
```

- Example of a data file

  /var/tmp/MasterFile01.csv

```
"01","1001","AAA","BlockA","1,000","1,000","Comment: Memo number 4023"
"02","1001","BBB","BlockB","","1,200","Comment: Memo number 4023"
"03","1002","CCC","BlockC","800","800","Comment: Memo number 4023"
```

- Example of a master definition file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<master xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="MASTER_01">
    <comment>Master definition_01</comment>
    <schemaFile>/var/tmp/SchemaFile01.csv</schemaFile>
    <dataFile>/var/tmp/MasterFile01.csv</dataFile>
    <skipHeader>false</skipHeader>
</master>
```

9.2.4 RDB Reference Definition File

This section explains the setup file for describing the RDB reference definition (RDB reference definition file).

The RDB reference definition file is an XML file that has "database" as the root element. An RDB reference definition file is created for each RDB server.

The items required in the RDB reference definition file depend on the RDB to be referenced.

Table 9.1 Symfoware Server referenced with the native interface

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of "database")	Development asset ID	ID that uniquely identifies the RDB reference definition. Used as the database name in complex event processing rules.	Up to 39 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
comment	Comment	Explanation of this definition.	Up to 1,000 characters.	Optional
dbName	Database name	Database name.	Up to 36 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
schema	Schema name	Schema name.	Up to 36 alphanumeric characters. Note: The first character must be a letter.	Mandatory
url	Database URL	Specify the following values: `hostNameOrIpAddress:portNumber` If the host name of the server to be connected is in FQDN (Fully Qualified Domain Name) format, specify the name excluding the domain name.	Up to 512 alphanumeric characters, hyphens (-), periods (.), or colons (:).	Mandatory
user	Access ID	Name of the user who will connect to the RDB server.	Up to 36 alphanumeric characters. Note: The first character must be a letter.	Mandatory
password	Access password	Password of the user who will connect to the RDB server. When the cepgetrsc command is used to reference definition information, this password is replaced by "*****" in the displayed output.	Up to 512 alphanumeric characters and the following symbols: `/%_#`	Mandatory
maxAge	Cache retention period	Time period (in seconds) for which RDB reference results are cached. If not caching results, omit the element, specify an empty value, or specify 0.	Integer from 0 to 2147483647	Optional
purgeInterval	Cache purge interval	Time interval (in seconds) at which a cache of RDB reference results will be flushed. A cache that is older than the cache retention period is flushed at this interval. If not caching results, omit the element or specify an empty value.	Integer from 1 to 2147483647	Optional

Table 9.2 Symfoware Server (Open Interface) and PostgreSQL

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of "database")	Development asset ID	ID that uniquely identifies the RDB reference definition. Used as the database name in complex event processing rules.	Up to 39 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
comment	Comment	Explanation of this definition.	Up to 1,000 characters.	Optional
jdbcClass	JDBC driver class	Class of the JDBC driver. Set the following value: `org.postgresql.Driver`	Up to 1,023 alphanumeric characters and symbols.	Mandatory

url	Database URL	URL of the database to be connected.	Up to 512 alphanumeric characters and the following symbols:
			`%:/?#[]@!`
			`$&'()`
			`*+,;-._~`
				Mandatory

user	Access ID	Name of the user who will connect to the RDB server.	Up to 512 characters.	Mandatory
password	Access password	Password of the user who will connect to the RDB server. When the cepgetrsc command is used to reference definition information, this password is replaced by "*****" in the displayed output.	Up to 512 characters.	Mandatory
maxAge	Cache retention period	Time period (in seconds) for which RDB reference results are cached. If not caching results, omit the element, specify an empty value, or specify 0.	Integer from 0 to 2147483647	Optional
purgeInterval	Cache purge interval	Time interval (in seconds) at which a cache of RDB reference results will be flushed. A cache that is older than the cache retention period is flushed at this interval. If not caching results, omit the element or specify an empty value.	Integer from 1 to 2147483647	Optional

Note

- If you use the cepgetrsc command to reference definition information, the access password will be replaced by "*****" in the displayed output. However, if you directly enter the access password in the database URL, the password is displayed on the screen. For security reasons, do not directly enter the access password in the database URL.

- If you specify the password in the RDB reference definition file, for security reasons you should delete the access password from the definition file after deployment to the CEP engine is complete, or alternatively delete the entire definition file.
- Even when backing up the RDB reference definition file, delete the access password from the definition file or encrypt the entire definition file to ensure that the password cannot be referenced.

Example

Definition examples of the RDB reference definition

Below are definition examples for connecting to the various RDB types. In each example, the results of referencing the database "MyDB" on the RDB server "RERDB01" are held in the cache for 1 hour (3600 seconds). The cache is checked every 10 minutes (600 seconds) to see if the retention period has elapsed.

Symfoware Server (referenced with the native interface) definition example

This example connects to a relational database using the schema name "user01", port number "26551", access ID "user01", and access password "bdcep".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<database xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RDBREF_01">
    <comment>RDB reference definition_01</comment>
    <dbName>MyDB</dbName>
    <schema>user01</schema>
    <url>RERDB001:26551</url>
    <user>user01</user>
    <password>bdcep</password>
    <maxAge>3600</maxAge>
    <purgeInterval>600</purgeInterval>
</database>
```

Symfoware Server (Open Interface) definition example

This example connects to a relational database using the port number "26500", access ID "user01", and access password "bdcep".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<database xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RDBREF_01">
    <comment>RDB reference definition_01</comment>
    <jdbcClass>org.postgresql.Driver</jdbcClass>
    <url>jdbc:postgresql://RERDB01:26500/MyDB</url>
    <user>user01</user>
    <password>bdcep</password>
    <maxAge>3600</maxAge>
    <purgeInterval>600</purgeInterval>
</database>
```

PostgreSQL definition example

This example connects to a relational database using the port number "26500", access ID "user01", and access password "bdcep".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<database xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="RDBREF_01">
    <comment>RDB reference definition_01</comment>
    <jdbcClass>org.postgresql.Driver</jdbcClass>
    <url>jdbc:postgresql://RERDB01:26500/MyDB</url>
    <user>user01</user>
    <password>bdcep</password>
    <maxAge>3600</maxAge>
    <purgeInterval>600</purgeInterval>
</database>
```

9.2.5 SOAP Listener Definition File

This section explains the setup file for describing the SOAP listener definition (SOAP listener definition file).

The SOAP listener definition file is an XML file that has "soapListener" as the root element. A SOAP listener definition file is created for each user-developed Web service.

The items to be set in the SOAP listener definition file are shown below:

Element or attribute	Item name	Explanation	Allowed values	Mandatory/ Optional
id (attribute of "soapListener")	Development asset ID	ID unique in the deployment CEP engine Used as the output adapter (SOAP listener) in complex event processing rules.	Up to 39 alphanumeric characters, underscores (_), or hyphens (-). Note: The first character must be a letter.	Mandatory
comment	Comment	Explanation of this definition.	Up to 1,000 characters.	Optional
url	Connection URL	Connection URL for the user-developed Web service.	Up to 512 alphanumeric characters and symbols.	Mandatory
nameSpace	Namespace	Namespace of the body of the SOAP message (SOAP body) to be sent to the user-developed Web service.	Up to 512 characters.	Mandatory
prefix	Namespace prefix	Namespace prefix to be used in the body of the SOAP message (SOAP body). When omitted, "ns" is used.	Up to 20 characters.	Optional
method	Root element	Root element name of the body of the SOAP message (SOAP body).	Up to 512 characters.	Mandatory

Note

Up to 32 SOAP listener definitions can be deployed to one CEP engine.

Example

Definition example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<soapListener xmlns="urn:xmlns-fujitsu-com:cspf:bdcep:v1" id="LISTEN_01">
    <comment>SOAP listener definition_01</comment>
    <url>http://192.168.11.249/WebServWAR/MyApp1Service</url>
    <nameSpace>http://webservice/</nameSpace>
    <prefix>ns</prefix>
    <method>cep</method>
</soapListener>
```

This example defines the notification of a message (event) that includes a SOAP body specifying that the root element is "`cep`" in the user-developed Web service with the connection destination URL "`http://192.168.11.249/WebServWAR/MyApp1Service`".

9.3 Setup Files for Terracotta Collaboration

This section explains the setup files required when Terracotta collaboration is used. The following two files are required for Terracotta collaboration:

- Terracotta Cache Configuration File
- Terracotta Collaboration Setup File

9.3.1 Terracotta Cache Configuration File

To use Virtual Data Window in order to use a Terracotta cache (known as Ehcache), you must place an Ehcache configuration file (ehcache.xml) on the CEP Server. Place the Ehcache configuration file in the following location:

```
/etc/opt/FJSVcep/config/ehcache.xml
```

Refer to the Terracotta manual for information on the Ehcache configuration file. The table below explains the settings required for Terracotta Collaboration.

Element or attribute						Description
ehcache						Root element of the configuration file.
	name					Specify the name of the cache manager specified when creating the cache.
	maxBytesLocalHeap					Size of the data pool to be used.
	terracottaConfig					Element for defining a Terracotta server.
		url				List of Terracotta servers in the format "*hostNameOrIpAddress:portNumber*", delimited with a comma (,).
	cache					Element for defining a cache. Multiple <cache> elements can be specified in a single <ehcache> element.
		name				Name of the cache. This is the cache name specified using vdw:ehcache.
		terracotta				Defined for using a Terracotta server.
			nonstop			Defined for use as a nonstop cache.
				immediateTimeout		Specify whether to respond with a timeout when a network disconnection is detected. Specify "true" as the value.
				timeoutMillis		Specify the standby time until timeout.
				timeoutBehavior		Specify the operation to be performed if a timeout occurs.
					type	Specify "exception" as the value.
		searchable				Defined for searching a cache.

Example

The following example uses the cache "`Cache001`" configured on two Terracotta servers (192.168.1.1 and 192.168.1.2) using Terracotta collaboration:

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"
  name="SearchConfig"
  maxBytesLocalHeap="64M">
    <terracottaConfig url="192.168.1.1:9510,192.168.1.2:9510"/>
    <cache name="Cache001">
        <terracotta>
            <nonstop immediateTimeout="true" timeoutMillis="3000">
                <timeoutBehavior type="exception"/>
            </nonstop>
        </terracotta>
        <searchable />
    </cache>
</ehcache>
```

Note

If you edit the file while the CEP engine is running, you must restart the CEP engine to enable the contents.

9.3.2 Terracotta Collaboration Setup File

Set information, such as the class path and the license key, to be used for Terracotta collaboration.

When you execute the cepstarteng command after creating this file, the class path is set and the CEP engine starts.

Create the Terracotta Collaboration setup file in /etc/opt/FJSVcep/config/terracotta_conf.

Format

Below is a definition example of the Terracotta Collaboration setup file. It comprises the lines described below.

Parameter definition line

A parameter definition line has the *parameterName=specifiedValue* format, where the value on the right side is assigned to the parameter on the left side. Tabs and spaces at the beginning and end of the line are ignored.

Note

Do not enter any spaces or tabs around the equals sign (=).

Blank line

A blank line consists of spaces and tabs only. The entire line is ignored by Terracotta Collaboration.

Parameters that can be set

Parameters that can be set are described below (all of them are mandatory):

Parameter name	Item name	Explanation	Allowed values	Mandatory/ Optional
BDCEP_TERRACOTTA_CLASSPATH	Terracotta Collaboration class path	Path of the Terracotta jar files used for Terracotta collaboration. Delimit multiple jar file paths with a colon (:).	Up to 1,023 alphanumeric characters and the following symbols: `/:-_.`	Mandatory
BDCEP_TERRACOTTA_LICENSEKEY	Terracotta license key	Path of the Terracotta license key.	Up to 1,023 alphanumeric characters and	Mandatory

Parameter name	Item name	Explanation	Allowed values	Mandatory/ Optional
			the following symbols: `/-_.`	

Example

Below is a definition example of the setup file when Terracotta is installed in /opt/bigmemory-max-4.0.1.

Refer to the Terracotta manual for information on the jar file to be specified for the Terracotta Collaboration class path.

```
BDCEP_TERRACOTTA_CLASSPATH=/opt/bigmemory-max-4.0.1/apis/ehcache/lib/ehcache-ee-2.7.1.jar:/opt/
bigmemory-max-4.0.1/apis/ehcache/lib/slf4j-api-1.6.6.jar:/opt/bigmemory-max-4.0.1/apis/ehcache/lib/
slf4j-jdk14-1.6.6.jar:/opt/bigmemory-max-4.0.1/common/lib/bigmemory-4.0.1.jar:/opt/bigmemory-
max-4.0.1/apis/toolkit/lib/terracotta-toolkit-runtime-ee-4.0.1.jar
BDCEP_TERRACOTTA_LICENSEKEY=/var/bigmemory/terracotta-license.key
```

Note

- Before using Terracotta Collaboration, the five jar files listed below must be set in the class path for Terracotta Collaboration.

  Note that the file names may differ depending on the Terracotta version and the environment, so check the file names in the version of Terracotta to be used before setting the jar files. Insert the version of each jar file in place of "*x*".

  - bigmemory-*x.x.x*.jar
  - ehcache-ee*x.x.x*.jar
  - slf4j-api-*x.x.x*.jar
  - slf4j-jdk14-*x.x.x*.jar
  - terracotta-toolkit-runtime-ee-*x.x.x*.jar

- If you edit the files while the CEP engine is running, you must restart the CEP engine to enable the contents.

9.4 Setup File for RDB Collaboration

This section explains the setup file required when RDB collaboration is used.

9.4.1 RDB Collaboration Setup File

Set the class path of the JDBC driver to be used for RDB collaboration.

When you execute the cepstarteng command after creating this file, the class path is set and the CEP engine starts.

Create the RDB Collaboration setup file in /etc/opt/FJSVcep/config/rdb_conf.

Format

Below is a definition example of the RDB Collaboration setup file. It comprises the lines described below.

Parameter definition line

A parameter definition line has the *parameterName=specifiedValue* format, where the value on the right side is assigned to the parameter on the left side. Tabs and spaces at the beginning and end of the line are ignored.

Note

Do not enter any spaces or tabs around the equals sign (=).

Blank line

A blank line consists of spaces and tabs only. The entire line is ignored by RDB Collaboration.

Parameters that can be set

Parameters that can be set are described below (all of them are mandatory):

Parameter name	Item name	Explanation	Allowed values	Mandatory/ Optional
BDCEP_RDB_CLASSPATH	RDB Collaboration class path	Set the jar file of the JDBC driver to be used for RDB collaboration. Delimit multiple jar files with a colon (:).	Up to 1,023 alphanumeric characters and the following symbols: `/:-_.`	Mandatory

Example

Below is a definition example of the setup file when Symfoware (Open Interface) is used as the relational database to be referenced.

```
BDCEP_RDB_CLASSPATH=/opt/symfoclient64/jdbc/lib/postgresql-jdbc4.jar
```

Note

- You need not set the class path if using Symfoware Server with the native interface as the relational database to be referenced.
- If referencing Symfoware Server (Open Interface) or PostgreSQL, set the JDBC driver in the class path. Refer to the manual for the relational database to be referenced for information on the jar files to be specified in the class path.
- If multiple jar files or class files with the same class name are set at the same time, the file set first has priority.
- If you edit the file while the CEP engine is running, you must restart the CEP engine to enable the contents.

9.5 Setting up for Installation

This section explains the setup file to be used at unattended installation of BDCEP (installation file).

9.5.1 Installation File

This section explains the format of and parameters that can be set in the installation file.

Format

Below is a definition example of the installation file - it consists of the line types below:

Parameter definition line

A parameter definition line has the *parameterName=specifiedValue* format, where the value on the right side is assigned to the parameter on the left side. Tabs and spaces at the beginning and end of the line or around the equals sign (=) are ignored.

Blank line

A blank line consists of spaces and tabs only. The entire line is ignored by the Installer.

Example

Below is a definition example of the installation file.

```
BDCEP_USER_NAME=isbdcep
BDCEP_GROUP_NAME=isbdcep
BDCEP_INITIAL_ENGINE_NAME=CepEngine
```

Parameters that can be set

Parameters that can be set are described below (all of them are mandatory):

Parameter name	Item name	Explanation	Allowed values	Mandatory/ Optional
BDCEP_USER_NAME	Engine execution user name	User name of the engine execution user.	Registered user name, up to 8 alphanumeric characters.	Mandatory
BDCEP_GROUP_NAME	Group name	Group name of the group to which the engine execution user belongs.	Registered group name, up to 8 alphanumeric characters.	Mandatory
BDCEP_INITIAL_ENGINE_NAME	Initial CEP engine name	Name of the CEP engine to be created at initial setup.	Up to 20 alphanumeric characters.	Mandatory

Note

Before executing unattended installation, ensure that the user and group specified in the installation file are registered in the system.

9.6 Characters Allowed in Item, Tag and Attribute Names

This section explains the characters that can be specified in item names, tag names, and attribute names.

9.6.1 For High-Speed Filter Rules and Master Definitions

For high-speed filter rules and master definitions, the characters that can be specified in CSV column item names, XML tag names, and attribute names are the single-byte characters shown below as well as multi-byte characters:

!	-	.	0	1	2	3	4	5	6	7	8	9	:
@	A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z	

Note

- A hyphen (-) or period (.) cannot be specified at the beginning of a line.
- A space ( ) cannot be specified (the bottom rightmost cell of the table above does not signify a space).

9.6.2 For Complex Event Processing Rules

For complex event processing rules, the characters that can be specified in CSV column item names, XML tag names, and attribute names have the following constraints:

- The first character must be a letter (a to z or A to Z).
- The second and subsequent characters must be letters (a to z or A to Z), digits (0 to 9), or underscores (_).
- The reserved words of complex event processing rules cannot be used, regardless of case.

  Refer to Chapter 1, "Complex Event Processing Language Reference" in the *Developer's Reference* for information on the reserved words of complex event processing rules.

Please note that the rules are case-sensitive.

9.7 CSV Format Supported

This section explains the CSV format supported.

Example

CSV example

```
"01","1001","AAA","BlockA","1,000","1,000","Comment: Memo number 4023"
"02","1001","BBB","BlockB","","1,200","Comment: Memo number 4023"
"03","1002","CCC","BlockC","800","800","Comment: Memo number 4023"
```

- Use commas (,) to delimit the items in the columns in one record.
- Records are separated by newline characters.
- Items can be enclosed by double quote marks (""). If a comma or newline needs to be specified in an item, enclose the item in double quote marks.
- If a double quote mark (") needs to be specified in the content of an item, enclose the item in double quote marks and then add a double quote mark before the one that is part of the item (that is, specify two consecutive double quote marks).

  Example

  When items 'A,AA', 'B*newline*BB', and 'C"CC' are included in a record

  ```
  "A,AA","B
  BB","C""CC"
  ```

- When data is enclosed by double quote marks, an error will occur if a character other than a comma (,) is used between items. Delete any excess characters around delimiting commas (,).

  Example

  Incorrect: There are unnecessary spaces (underlined) between items (around the comma).

  ```
  "AAA",_"BlockA"
  ```

Correct: There are no unnecessary spaces between items.

```
"AAA","BlockA"
```

- The maximum size of one record is 32 MB (megabytes).
- For input events, records do not need to be separated by newlines. Multiple records cannot be handled in one input event.

Glossary

Apache Hadoop

Open-source Hadoop software developed by the Apache Software Foundation (ASF).

BDCEP (Interstage Big Data Complex Event Processing Server)

Fujitsu software for analyzing and assessing massive volumes of event data in real time by using the complex event processing technology.

BDPP (Interstage Big Data Parallel Processing Server)

Fujitsu software for processing massive data volumes by using the Apache Hadoop with Fujitsu proprietary technology incorporated.

Cache (RDB Collaboration)

In complex event processing, a feature that temporarily accumulates queries referenced using RDB Collaboration and their results in a CEP engine and processes previously referenced queries at high speed.

CEP (Complex Event Processing)

Refer to "complex event processing".

CEP engine

Refer to "high-performance CEP engine".

CEP Server

The server on which BDCEP runs.

CEP service

Service that runs on the CEP Server and manages the CEP engine.

complex event processing

Technique of rapidly analyzing, assessing and processing in real time the content and status of massive volumes of event data being sent continuously.

It is used in the Complex Event Processing feature provided by BDCEP.

complex event processing language

Language for specifying the rules of complex event processing.

It allows the use of SQL format.

complex event processing rule

Rule described using the complex event processing language, consisting of at least one complex event processing statement.

complex event processing statement

Structural unit of rules specified using the complex event processing language.

A complex event processing statement describes processing using statements such as the SELECT statement and the CREATE WINDOW statement.

Custom Listener

Feature that processes the results of a complex event processing statement by using a user-developed Java class.

Debug Listener

Feature that outputs debug messages to the engine log when a complex event processing statement is executed.

engine execution user

User of the operating system when a CEP engine is running.

engine log

File containing the output of CEP engine system messages and debug messages defined in the rules.

event data

Time-series data, such as sensing data or location information about people, and that shows when something occurred.

event log

File containing event data which is recorded by Logging.

event log analysis application

Application that analyzes event logs in the Hadoop system.

event sender application

Application that sends event data to a CEP engine.

event type definition

Definition information about the format of event data.

The event data can be specified in XML or CSV format.

filter language

Refer to "high-speed filter language".

filter rule

Refer to "high-speed filter rule".

filter statement

Refer to "high-speed filter statement".

Hadoop

Technique for effectively performing distributed and parallel processing of data accumulated in massive volumes. It basically consists of the HDFS (Hadoop Distributed File System) and the MapReduce which is known as a parallel distributed processing technique.

Refer to "Apache Hadoop".

Hadoop Collaboration

In BDCEP Logging context, a feature for recording event data from the CEP Server directly in the Hadoop system.

Hadoop system

System on which Hadoop is running.

HDFS (Hadoop Distributed File System)

Distributed file system used by Hadoop.

For the placement of big data, HDFS creates multiple replicas of data blocks and distributes them on nodes called DataNodes, which are then managed by nodes called NameNodes.

high-performance CEP engine

The processing unit in BDCEP.

It allows massive volumes of event data to be processed rapidly by combining High-speed Filter with conventional complex event processing.

High-speed Filter

Feature that uses a technique unique to Fujitsu to allow high-speed filtering of massive volumes of event data while matching the event data with the master data.

high-speed filter language

Language for specifying the rules in High-speed Filter.

It allows the use of IF-THEN format.

high-speed filter rule

Rule described using the high-speed filter language, consisting of at least one high-speed filter statement.

high-speed filter statement

Structural unit of rules specified using the high-speed filter language.

A high-speed filter statement is described for each event type definition.

HTTP adapter

Input Adapter that allows event data to be received using HTTP communication.

It allows more lightweight communication when compared to a SOAP adapter.

initial CEP engine

CEP engine created at installation.

Input Adapter

Feature that receives event data from outside the CEP Server.

Logging

Feature that records event data in a Hadoop system or in the engine log.

Logging Listener

Feature that Logging the processing results of a complex event processing statement.

master data

Data used by BDCEP to match event data in High-speed Filter processing.

It is deployed in the CEP Server in CSV format, and consists of schema files (where the item names for each column are defined), and data files (where the data rows are stored).

master definition

Definition information of the master data to be referenced during High-speed Filter processing.

Output Adapter

Feature that sends the processing results of complex event processing outside the CEP Server.

RDB reference definition

RDB connection information used with RDB Collaboration.

RDB Collaboration

In complex event processing provided by BDCEP, a feature that references data stored in a relational database that is external to the CEP Server.

resource log

Log containing the resource usage of a CEP engine.

It is stored in CSV format, so it can be analyzed using a tool such as Excel.

rule definition

Processing details for the High-speed Filter (high-speed filter rule) and complex event processing (complex event processing rule) features.

sensing data

Data sent from a variety of sensors.

Sensing data is a type of event data.

SOAP (Simple Object Access Protocol)

Lightweight protocol, which is subject to ongoing standardization work by the World Wide Web Consortium (W3C).

For its communication infrastructure, SOAP uses the Internet standards of Hyper Text Transfer Protocol (HTTP) and (eXtensible Markup Language (XML).

SOAP adapter

Input Adapter that allows event data to be received using SOAP.

SOAP Listener

Feature that uses SOAP to send the processing results of a complex event processing statement to an external system.

SOAP listener definition

Defines information required to perform communication using a SOAP Listener.

It defines information such as the URL of the external system that is to be the communication target.

social media

Services and applications that provide open communication generated by diverse content (such as text, voice and video) exchanged and shared on the Internet by a variety of people, in contrast to the traditional information media (such as newspapers and television).

socket adapter

Input Adapter that performs communication using a format unique to BDCEP.

Socket adapters are more suitable for receiving massive volumes of event data than a SOAP adapter or HTTP adapter.

Terracotta application

Client application in Interstage Terracotta BigMemory Max. It updates data in a cache by, for example, storing initial data in the cache on a Terracotta server.

Terracotta server

Terracotta server in Interstage Terracotta BigMemory Max.

Terracotta Collaboration

In complex event processing provided by BDCEP, a feature that references the cache of a Terracotta server external to the CEP Server. It is used for referencing frequently updated data from a complex event processing rule.

user-developed Java class

Java class that runs when event data is passed from the custom listener.

user-developed Web service

Application that uses SOAP to receive processing results of a complex event processing statement sent from a SOAP Listener.

WSDL (Web Services Description Language)

Interface description language in Web services, which allows the description of information such as the access point in Web services (URL), the protocol to be used (SOAP, HTTP, or MIME), and the message format (XML Schema).

It is used in the development of a user-developed Web service, which is an application for collaborating with BDCEP.