HITACHI
Inspire the Next

uCosminexus Stream Data Platform - Application Framework

Description

3020-3-V01(E)

■ Relevant program products

P-2464-9B17 uCosminexus Stream Data Platform - Application Framework 01-00 (for Windows Server 2008)

■ Trademarks

BSAFE is a registered trademark or a trademark of EMC Corporation in the United States and/or other countries.

Internet Explorer is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

RSA is a registered trademark or a trademark of EMC Corporation in the United States and/or other countries.

Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows Server is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. The original software is available from ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/

This product includes software developed by Andy Clark.

This product includes software developed by Ben Laurie for use in the Apache-SSL HTTP server project.

This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (http://relaxngcc.sf.net/).

This product includes software developed by IAIK of Graz University of Technology.

This product includes software developed by Ralf S. Engelschall <rse@engelschall.com> for use in the mod_ssl project (http://www.modssl.org/).

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (http://java.apache.org/).

This product includes software developed by the University of California, Berkeley and its contributors.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

uCosminexus Stream Data Platform - Application Framework contains RSA(R) BSAFE^{TM} software from EMC Corporation.

HITACHI
Inspire the Next

Hitachi, Ltd.

Other product and company names mentioned in this document may be the trademarks of their respective owners. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Full name or meaning	Abbreviation
Microsoft(R) Windows Server(R) 2008 Enterprise	Windows Server 2008 or Windows
Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit	
Microsoft(R) Windows Server(R) 2008 R2 Enterprise	
Microsoft(R) Windows Server(R) 2008 R2 Standard	
Microsoft(R) Windows Server(R) 2008 Standard	
Microsoft(R) Windows Server(R) 2008 Standard 32-bit	
Microsoft(R) Windows(R) Internet Explorer(R)	Internet Explorer

■ Restrictions

Information in this document is subject to change without notice and does not represent a commitment on the part of Hitachi. The software described in this manual is furnished according to a license agreement with Hitachi. The license agreement contains all of the terms and conditions governing your use of the software and documentation, including all warranty rights, limitations of liability, and disclaimers of warranty.

Material contained in this document may describe Hitachi products not available or features not available in your country.

Printed in Japan.

■ Edition history

Aug. 2011: 3020-3-V01(E)

■ Copyright

Preface

This manual provides an overview and a basic understanding of uCosminexus Stream Data Platform - Application Framework. It is intended to provide an overview of the features and system configurations of uCosminexus Stream Data Platform - Application Framework, and to give you the basic knowledge needed to set up and operate such a system.

Intended readers

This manual is intended for all users of uCosminexus Stream Data Platform - Application Framework.

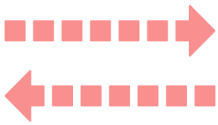
Readers of this manual must have:

- A basic knowledge of operating systems

Conventions: Diagrams

This manual uses the following conventions in diagrams:

- Flow of stream data



Conventions: Fonts and symbols

The following table explains the fonts used in this manual:

Font	Convention
Bold	**Bold** type indicates text on a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.

Font	Convention
Italics	*Italics* are used to indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` *Italics* are also used for emphasis. For example: • Do *not* delete the configuration file.
`Code font`	A `code font` indicates text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`

Conventions: Version numbers

The version numbers of Hitachi program products are usually written as two sets of two digits each, separated by a hyphen. For example:

- Version 1.00 (or 1.0) is written as 01-00.
- Version 2.05 is written as 02-05.
- Version 2.50 (or 2.5) is written as 02-50.
- Version 12.25 is written as 12-25.

The version number might be shown on the spine of a manual as *Ver. 2.00,* but the same version number would be written in the program as *02-00*.

Contents

Chapter

1. What is Stream Data Platform - AF?

Stream Data Platform - Application Framework (AF) is a product that enables you to process stream data; that is, it allows you to analyze in real-time large sets of data as they are being created. This chapter provides an overview of Stream Data Platform - AF and explains its features. This chapter also gives an example of adding Stream Data Platform - AF to your current workflow, and it describes the system configuration needed to set up and run Stream Data Platform - AF.

1.1 A data processing system that analyzes the "right now"

Our societal infrastructure has been transformed by the massive amounts of data being packed into our mobile telephones, IC cards, home appliances, and other electronic devices. As a result, the amount of data handled by data processing systems continues to grow daily. The ability to quickly summarize and analyze this data can provide us with valuable new insights. To be useful, any real-time data processing system must have the ability to create new value from the massive amounts of data that is being created every second.

Stream Data Platform - AF responds to this challenge by giving you the ability to perform *stream data processing*. Stream data processing gives you real-time summary analysis of the large quantities of time-sequenced data that is always being generated, as soon as the data is generated.

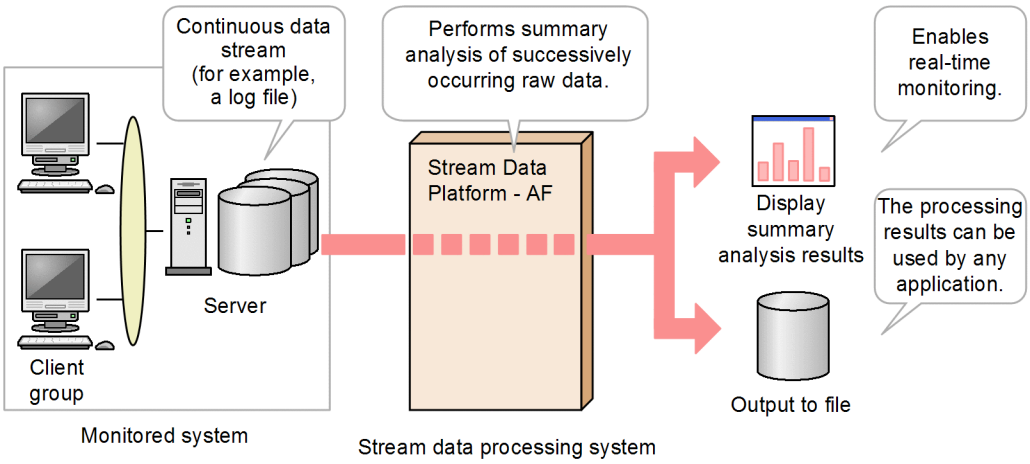
For example, think how obtaining real-time summary information on what was searched for from peoples PCs and mobile phones could increase your product sales opportunities. If a particular product becomes a hot topic on product discussion sites, you expect the demand for it to increase, so more people would tend to search for that product on the various search sites. You can identify such products by using stream data processing to analyze the number of searches in real-time and provide summary results. This information allows retail outlets to increase their orders for the product before the demand hits, and for the manufacturer to quickly ramp up production of the product.

On the IT systems side, demand for higher operating efficiencies and lower costs continues to grow. At the same time, the increasing use of virtualization and cloud computing results in ever larger and more complex systems, making it even more difficult for IT to get a good overview of their system's state of operation. This means that it often takes too long to detect and resolve problems when they occur. Now, by using stream data processing to monitor the operating state of the system in real-time, a problem can be quickly dealt with as soon as it occurs. Moreover, by analyzing trends and correlations in the information about the system's operations, warning signs can be detected, which can be used to prevent errors from ever occurring.

Adding Stream Data Platform - AF to your data processing system gives you a tool that is designed for processing these large volumes of data.

The following figure provides an overview of a configuration that uses Stream Data Platform - AF to implement stream data processing.

Figure 1-1: Overview of a stream data processing configuration that uses Stream Data Platform - AF



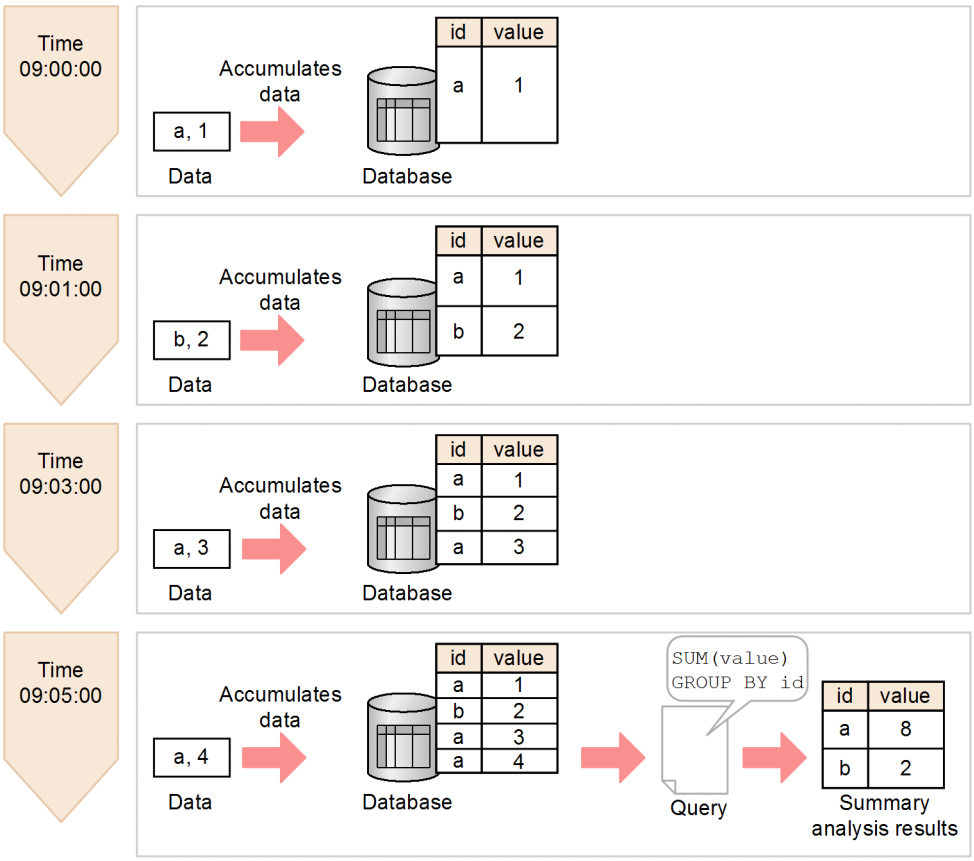
Introducing Stream Data Platform - AF into your stream data processing system allows you to perform summary analysis of data as it is being created.

For example, by using a stream data processing system to monitor system operations, you can summarize and analyze log files output by a server and HTTP packets sent over a network. These results can then be displayed on the dashboard, allowing you to monitor your system's operations in real-time. In this way, you can quickly resolve system problems as they occur, improving operation and maintenance efficiencies. You can also store the processing results in a file, allowing you to use other applications to further review or process the results.

To give you a better idea of how stream data processing carries out real-time processing, stream data processing is compared to conventional stored data processing in the following example.

Figure 1-2 shows conventional stored data processing.

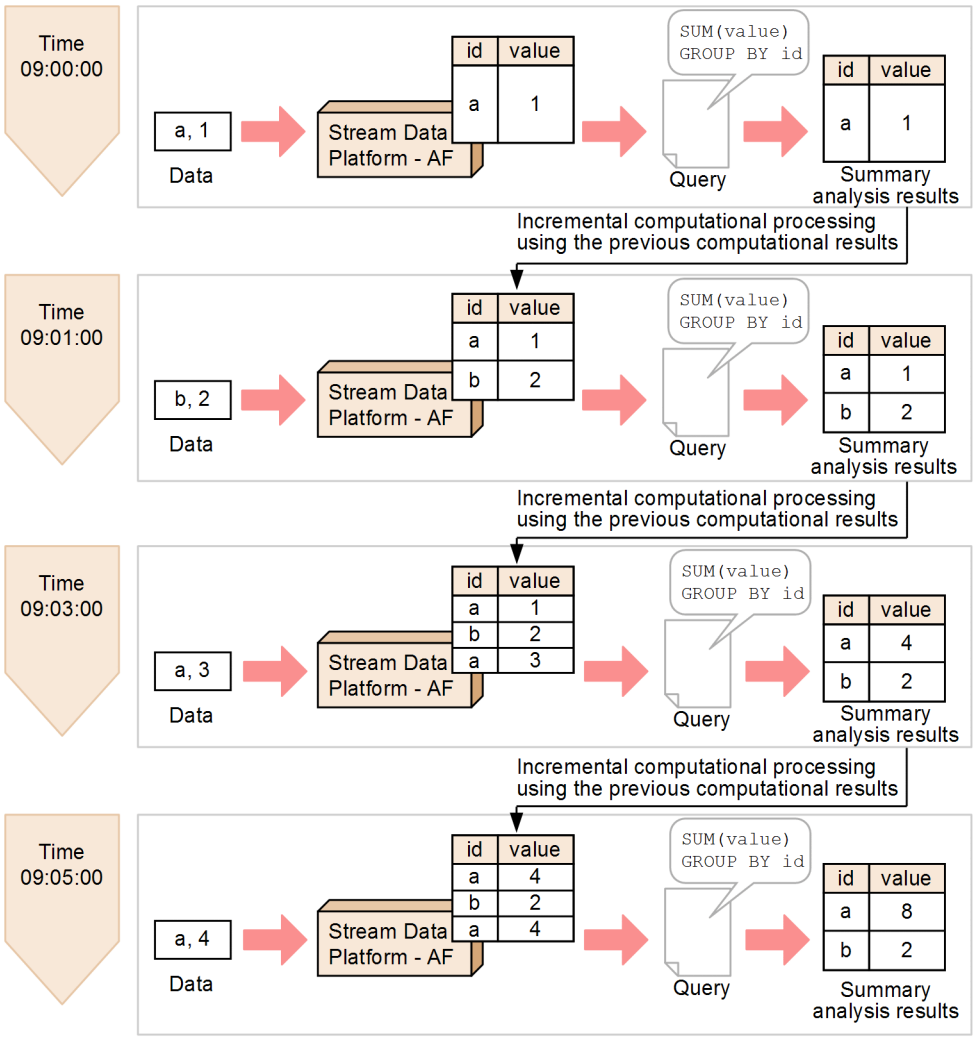
Figure 1-2: Stored data processing



Data processed using stored data begins by storing the data sequentially in a database as it occurs. Processing is not actually performed until a user issues a query for the data stored in the database, and summary analysis results are returned. Because data that is already stored in a database is searched when the query is received, there is a time lag between the time the data is collected and the time the data summary analysis results are produced. In the figure, processing of data that was collected at 09:00:00 is performed by a query issued at 09:05:00, obviously lagging behind the time the data was collected.

Figure 1-3 shows stream data processing.

Figure 1-3: Stream data processing



With stream data processing, you pre-load a query (summary analysis scenario) that will perform incremental data analysis, thus minimizing the amount of computing that is required. Moreover, because data analysis is triggered by the data being input, there is no time lag between it and the time the data is collected, providing you with real-time data summary analysis. This kind of stream data processing, in which processing is triggered by the input data itself, is a superior approach for data that is generated sequentially.

Therefore, the ability to perform stream data processing that you gain by integrating Stream Data Platform - AF into your system allows you to get a real-time summary and analysis of the data.

1.2 Stream Data Platform - AF features

Stream Data Platform - AF has the following features:

- High-speed processing of large sets of time-sequenced data
- Summary analysis scenario definitions that require no programming

The following subsections explain these features.

1.2.1 High-speed processing of large sets of time-sequenced data

Stream Data Platform - AF uses both *in-memory processing* and *incremental computational processing*, which allows it to quickly process large sets of time-sequenced data.

In-memory processing

With in-memory processing, data is processed while it is still in memory, thus eliminating unnecessary disk access.

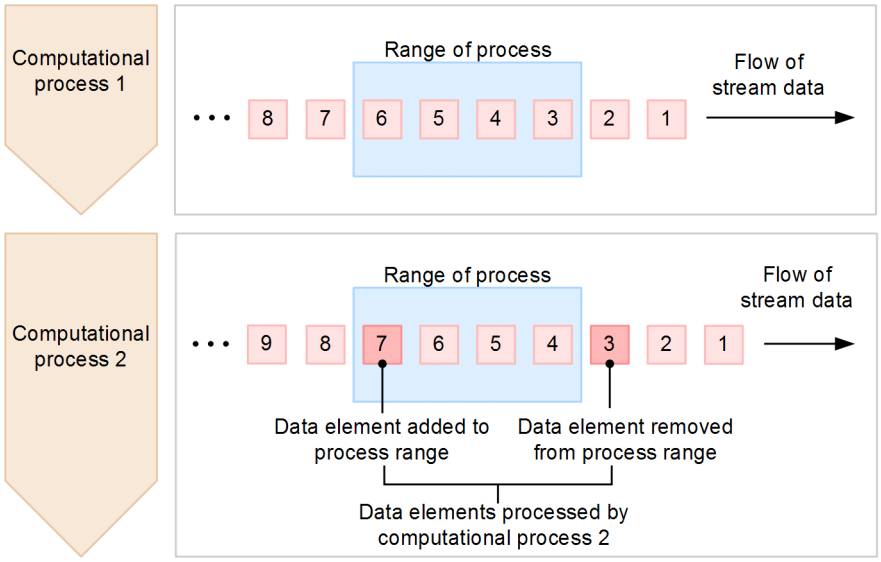
When processing large data sets, the time required to perform disk I/O can be significant. By processing data while it is still in memory, Stream Data Platform - AF avoids excess disk I/O, enabling data to be processed faster.

Incremental computational processing

With incremental computational processing, a pre-loaded query is processed iteratively when triggered by the input data, and the processing results are available for the next iteration. This means that the next set of computations does not need to process all of the target data elements; only those elements that have changed need to be processed.

The following figure shows incremental computation on stream data as performed by Stream Data Platform - AF.

Figure 1-4: Incremental computation performed on stream data



As shown in the figure, when the first stream data element arrives, Stream Data Platform - AF performs computational process 1. When the next stream data element arrives, computational process 2 simply removes data element 3 from the process range and adds data element 7 to the process range, building on the results of computational process 1. This minimizes the total processing required, thus enabling the data to be processed faster.

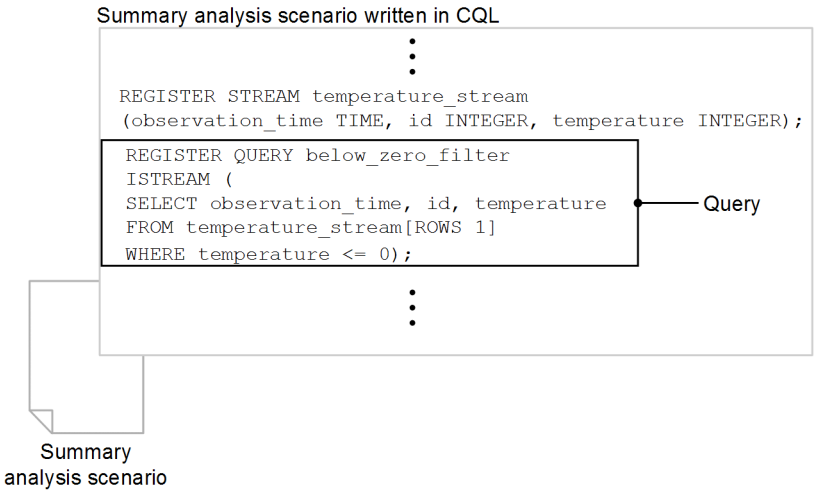
1.2.2 Summary analysis scenario definitions that require no programming

The actions performed in stream data processing are defined by queries that are called *summary analysis scenarios*. Definitions for these summary analysis scenarios are written in a language called *CQL*, which is very similar to SQL, the standard language used to manipulate databases. This means that you do not need to create a custom analysis application to create summary analysis scenarios. Summary analysis scenarios can also be modified simply by changing the definition files written in CQL.

Stream data processing actions written in CQL are called queries. In a single summary analysis scenario, multiple queries can be coded.

For example, the following figure shows a summary analysis scenario written in CQL for a temperature monitoring system that has multiple observation sites, each with an assigned ID. The purpose of the query is to summarize and analyze all of the below freezing point data found in the observed data set.

Figure 1-5: Example of using CQL to write a summary analysis scenario



CQL is a general-purpose query language that can be used to specify a wide range of processing. By combining multiple queries, you can define summary analysis scenarios to handle a variety of operations.

1.3 Example of Stream Data Platform - AF in use

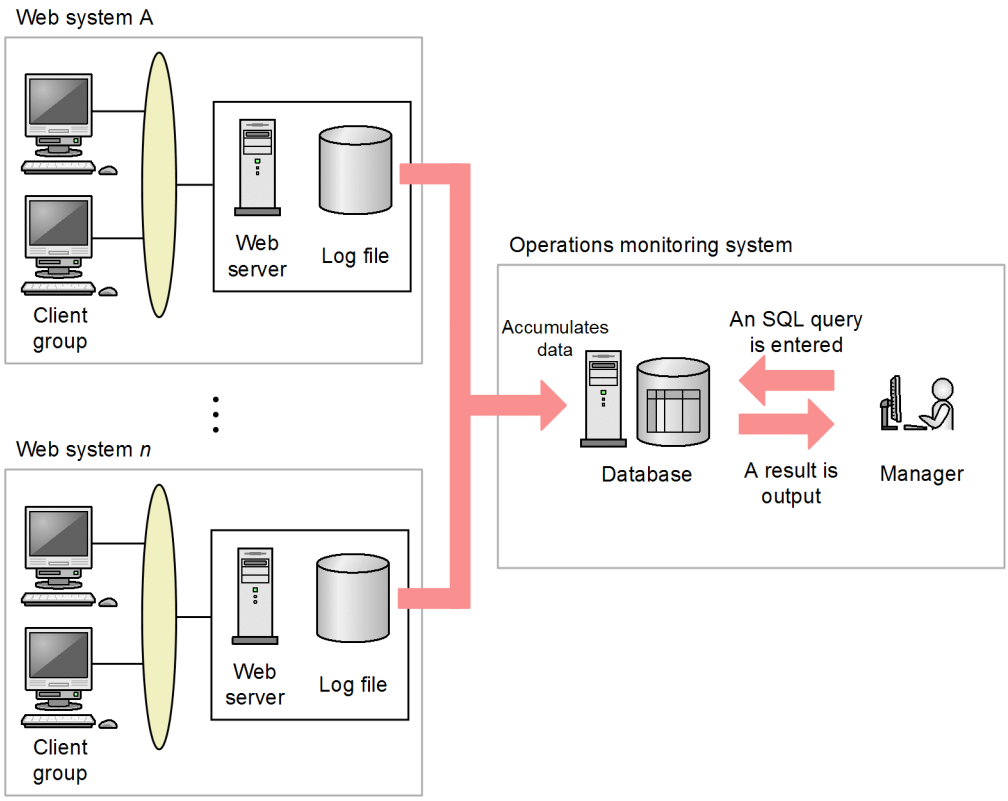
Stream Data Platform - AF is able to summarize and analyze a variety of data. This section presents an example of monitoring the operation of Web systems by summary analysis of HTTP packets.

Reference note:

You can also use summary analysis of log files to monitor system operation.

By using Stream Data Platform - AF to perform real-time monitoring of HTTP packets, you can keep track of system operations on a regular basis through actions such as checking the log files of each Web server in the Web system. The following figure shows the configuration of the system explained in this section.

Figure 1-6: Example of a system configuration for operations monitoring of Web systems



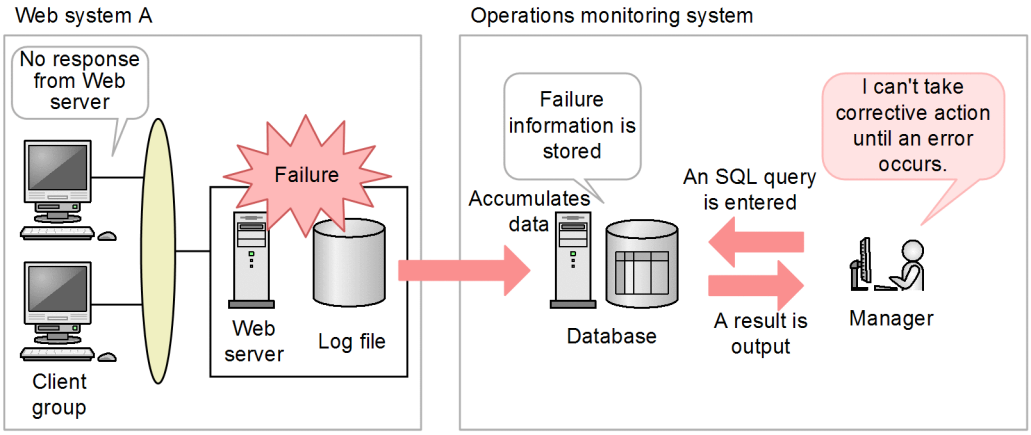
As shown in the above figure, after the Web server log files are stored in the database, a manager using the operations monitoring system can keep track of all of the Web systems on a regular basis by entering SQL queries that summarize and analyze the stored data.

The following paragraphs explain the differences in operation monitoring with and without the use of Stream Data Platform - AF for a problem that occurs in Web system A.

Without Stream Data Platform - AF

If a failure occurs on a Web server, no corrective action is taken unless the manager happens to check the log file and perform a summary analysis. Because failures cannot be detected in real-time, fixes cannot be applied until long after the problem occurs. The following figure illustrates this situation.

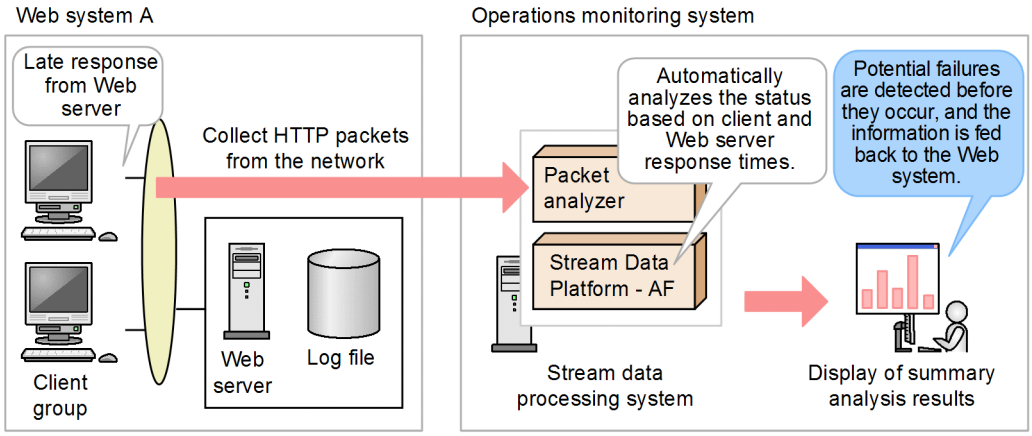
Figure 1-7: Example of operations monitoring of a Web system (without Stream Data Platform - AF)



With Stream Data Platform - AF

Once Stream Data Platform - AF is installed, information about the amount of data traffic carried over the network and its fluctuations can be acquired directly from HTTP packets before the information is ever written to a log file. This allows the manager to monitor the operating state of the system in real-time, and to detect potential failures before they occur, thus improving maintenance efficiency. A packet analyzer (a program that collects HTTP packets carried over a network) is used to give Stream Data Platform - AF the information it needs to summarize and analyze HTTP packets. The following figure illustrates this situation.

Figure 1-8: Example of operation monitoring of a Web system (with Stream Data Platform - AF)



Stream Data Platform - AF looks at the HTTP packets carried over the network, and analyzes system status in real-time based on the responses between the clients and the Web server. For example, by analyzing the time a Web server takes to respond to a request from a client, the manager can quickly detect the beginning of a slowdown in system response time. By using this information to adjust the Web system, the manager can prevent a failure from occurring.

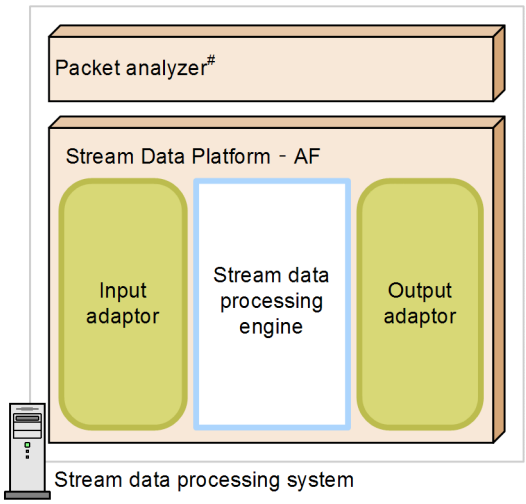
1.4 Stream Data Platform - AF system configuration and prerequisite programs

This section describes how to configure a stream data processing system that uses Stream Data Platform - AF and the prerequisite programs that are required depending on the system objectives.

1.4.1 System configuration

The following figure shows the configuration of a stream data processing system that uses Stream Data Platform - AF.

Figure 1-9: Configuration of a stream data processing system



#
Required to input HTTP packets.

The following paragraphs describe the components shown in this figure that make up Stream Data Platform - AF.

- Input adaptor

  An input adaptor converts input data into a form that the stream data processing engine can handle, performs other operations such as filtering the data, and then sends the results to the stream data processing engine.

  There are two types of input adaptors: standard adaptors and custom adaptors. Using a standard adaptor, you can input data such as files and HTTP packets output by the system being analyzed. By using a custom adaptor, you can input a wide variety of data, depending on the applications that you create.

For details about input adaptors, see *3. Exchanging Data*.

- Stream data processing engine

  The stream data processing engine processes data received from the input adaptor according to pre-loaded queries. For details about the stream data processing engine, see *2. Stream Data Processing*.

- Output adaptor

  The output adaptor receives data processed by the stream data processing engine, filters the data and converts it to a specified format, and then outputs it to an output destination.

  There are two types of output adaptors: standard adaptors and custom adaptors. Using a standard adaptor, you can output the processing results to a file or to the dashboard. By using a custom adaptor, you can output data to a wide variety of output destinations, depending on the applications that you create.

  For details about output adaptors, see *3. Exchanging Data*.

1.4.2 Prerequisite programs

Depending on the data that you want to read and/or write, Stream Data Platform - AF might require prerequisite programs. You do not need the prerequisite programs listed below unless you plan to input or output data as explained in this subsection.

(1) Inputting HTTP packets

To use Stream Data Platform - AF to summarize and analyze HTTP packets input from a network, you need a packet analyzer capable of acquiring packets in the Pcap format.

Stream Data Platform - AF supports the packet analyzer listed below:

- WinDump

(2) Outputting data to the dashboard

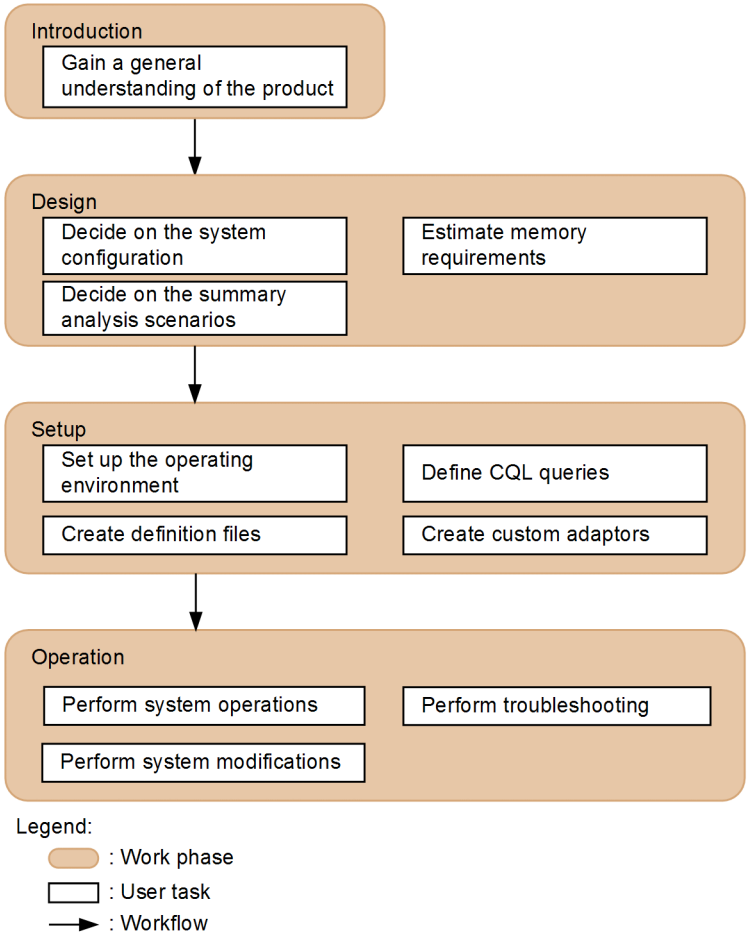
To use the dashboard to output and display the summary analysis results that Stream Data Platform - AF generates from stream data, you need the following programs:

- Internet Explorer
- Flash Player

1.5 Workflow beginning with the introduction of Stream Data Platform - AF

The following figure shows the general workflow beginning with the introduction of Stream Data Platform - AF.

Figure 1-10: Workflow beginning with the introduction of Stream Data Platform - AF



The following paragraphs provide an overview of the work phases shown in the figure.

- Introduction

  In the introduction phase, you gain a general understanding of the Stream Data

Platform - AF product before you begin using it.

- Design

  In the design phase, you design the system into which Stream Data Platform - AF will be introduced. It includes such tasks as determining the system configuration and summary analysis scenarios, and estimating memory requirements.

- Setup

  In the setup phase, you configure the system in which Stream Data Platform - AF has been introduced. It includes such tasks as setting up the operating environment and creating definition files. It also includes defining CQL queries.

  To create custom adaptors, you use the APIs provided with Stream Data Platform - AF.

- Operation

  In the operation phase, you perform operations on the system in which Stream Data Platform - AF has been introduced, and modify operations as necessary. You also perform troubleshooting (collect failure information, take corrective action, and so on) if a problem occurs.

Read this manual to gain a general product overview. Read the other manuals in this series for details about the various work phases.

For details about the correspondence between the tasks and the manuals in this series, see *4.1 Correspondence between user tasks and the manuals in this series*.

Chapter

2. Stream Data Processing

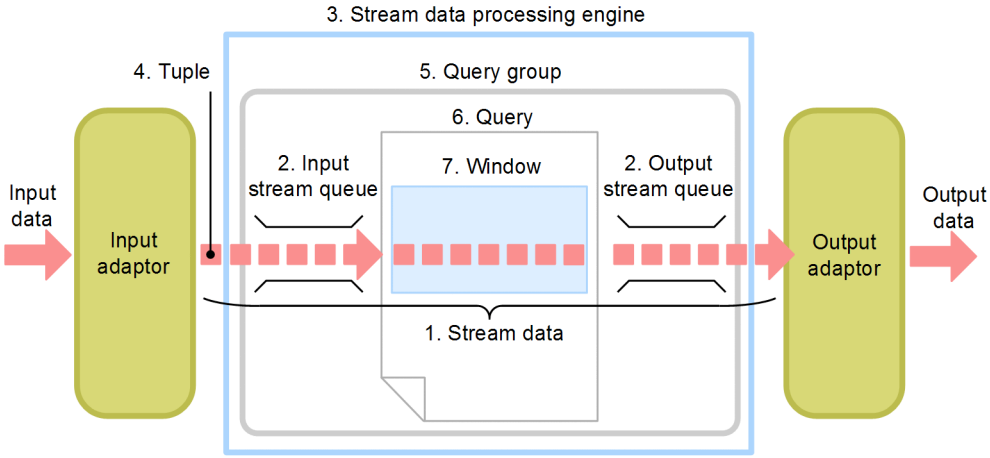
This chapter describes the components used in stream data processing, and the CQL query language used to define summary analysis scenarios.

2.1 Components used in stream data processing

This section describes the components used in stream data processing.

The following figure shows the components used in stream data processing.

Figure 2-1: Components used in stream data processing



This section explains the following components shown in the figure.

1. Stream data

   Large quantities of time-sequenced data that is continuously generated

2. Input and output stream queues

   Parts of the stream data path

3. Stream data processing engine

   The part of the stream data processing system that actually processes the stream data

4. Tuple

   A stream data element that consists of a combination of two or more data values, one of which is a time (timestamp)

5. Query group

   A summary analysis scenario used in stream data processing. Different query groups are created for different operational objectives.

6. Query

   The action performed in stream data processing. Queries are written in CQL.

7. Window

   The target range of the stream data processing. The amount of stream data that is included in the window is the process range. It is defined in the query.

For details about the input and output adaptors, see *3. Exchanging Data*.

2.1.1 Stream data

Stream data refers to large quantities of time-sequenced data that is continuously generated.

Stream data flows based on the stream data type (*STREAM*) defined in CQL, enters through the input stream queue, and is processed by the query. The query's processing results are converted back to stream data, and then passed to the output stream queue and output.

2.1.2 Input and output stream queues

The *input stream queue* is the path through which the input stream data is received. The input stream queue is coded in the query using CQL statements for reading streams. For details about the CQL code used to define streams, see *2.2.1(1) Defining a stream (REGISTER STREAM clause)*.

The *output stream queue* is the path through which the processing results (stream data) of the stream data processing engine are output. The output stream queue is coded in the query using CQL statements for outputting stream data. For details about the CQL code used to output the processing results of a query as stream data, see *2.2.2(3) Stream operations (outputting the data processing results)*.

The type of stream data that passes through the input stream queue is called an *input stream*, and the type of stream data that passes through the output stream queue is called an *output stream*.

2.1.3 Stream data processing engine

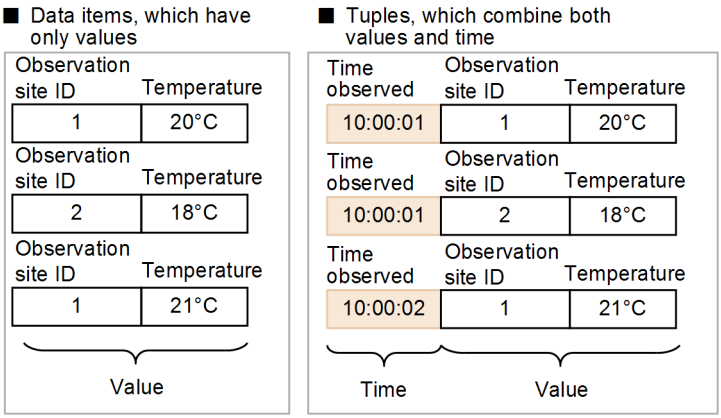
The *stream data processing engine* is the main component of Stream Data Platform - AF and actually processes the stream data. The stream data processing engine performs real-time processing of stream data sent from the input adaptor, according to the definitions in a pre-loaded query. It then outputs the processing results to the output adaptor.

2.1.4 Tuple

A *tuple* is a stream data element that consists of a combination of data values and a time value (*timestamp*).

For example, for temperatures observed at observation sites 1 (ID: 1) and 2 (ID: 2), the following figure compares data items, which have only values, with tuples, which combine both values and time.

Figure 2-2: Comparison of data items, which have only values, with tuples, which combine both values and time



By setting a timestamp indicating the observation time to each tuple as shown in the figure, data can be processed as stream data, rather than handled simply as temperature information from each observation site.

There are two ways to set the tuple's timestamp: the *server mode* method, where the timestamp is set based on the time the tuple arrives at the stream data processing engine, and the *data source mode* method, where the timestamp is set at the time that the data was generated. Use the data source mode when you want to process stream data sequentially based on the time information in the data source, such as when you perform log analysis.

The following subsections explain each mode.

Reference note:

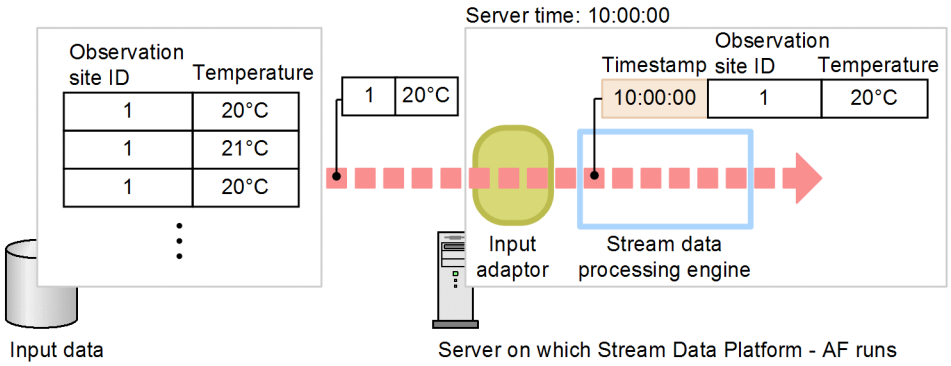
The tuples input to the stream data processing engine and the tuples output from the stream data processing engine are both output to a log file (*tuple log*). You can use the tuple log to re-execute a query or to check processing results. For details about the tuple log, see the *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*.

(1) Setting the server system time to the tuples (server mode)

The mode in which the tuple's timestamp is set to the system time of the server on which Stream Data Platform - AF runs when the tuple is read by the stream data processing engine is called the *server mode*. The following figure shows a timestamp

being set in the server mode.

Figure 2-3: A timestamp being set in the server mode

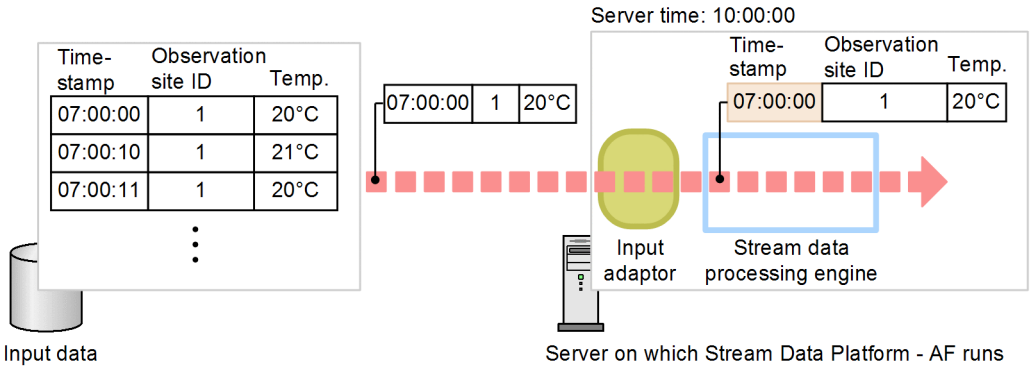


In the server mode, Stream Data Platform - AF assigns the server time to each tuple, regardless of whether the input data already has a timestamp.

(2) Using the data source time in the tuples (data source mode)

If time information is being written in the data by the data source from which data is being collected, the mode in which that time information in the tuples is used is called the *data source mode*. The following figure shows a timestamp being set in the data source mode.

Figure 2-4: A timestamp being set in the data source mode



Legend:
Temp.: Temperature

In the data source mode, the timestamp already in the input data is used for each tuple.

Note that, if you perform stream data processing in the data source mode, the data has to be queued in the order specified by the timestamps. If the potential exists for the

stream data to contain a timestamp error, see the information about how to queue timestamps in order in the *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*.

2.1.5 Query group

A *query group* is a summary analysis scenario for stream data that has already been created by the user. A query group consists of an input stream queue (input stream), an output stream queue (output stream), and a query.

You create and load query groups to accomplish specific operations. You can load any number of query groups.

2.1.6 Query

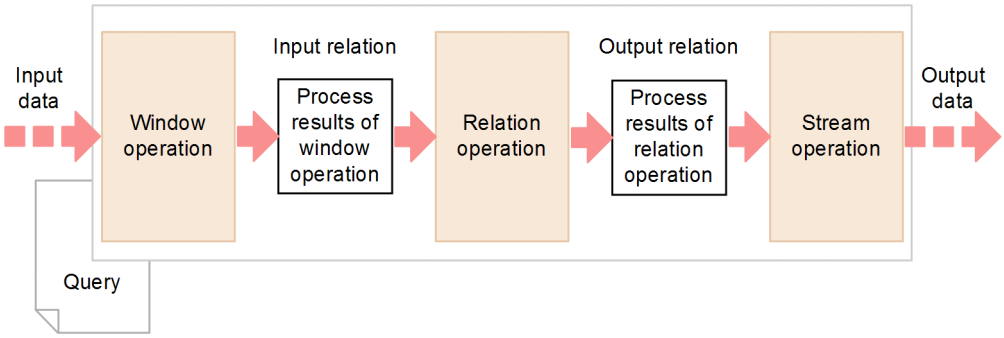
A *query* defines the processing that is performed on stream data. Queries are written in a *query definition file* using CQL. For details about the query definition file, see the *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*.

Queries define the following three types of operations:

- *Window operations*, which retrieve the data to be analyzed from the stream data
- *Relation operations*, which process the retrieved data
- *Stream operations*, which convert and output the processing results

The following figure shows the relationship between these operations.

Figure 2-5: Relationship between the operations defined by a query



A window operation retrieves stream data elements within a specific time window. The data gathered in this process (tuple group) is called an *input relation*.

A relation operation processes the data retrieved by the window operation. The tuple group generated in this process is called an *output relation*.

A stream operation takes the data that was processed by the relation operation,

converts it to stream data and outputs it.

For details about each of these operations, see *2.2.2 Using data manipulation CQL to specify operations on stream data*.

Stream data is processed according to the definitions in the query definition file used by the stream data processing engine. For details about the contents of a query definition file, see *2.2 Using CQL to process stream data*. For query code examples, see *2.3 Implementation examples of using CQL to process stream data*.

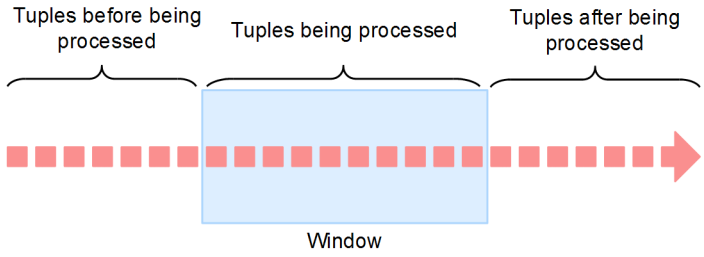
2.1.7 Window

A *window* is a time range set for the purpose of summarizing and analyzing stream data. It is defined in a query.

In order to summarize and analyze any data, you must clearly define a target scope. With stream data as well, you must first decide on a fixed range, and then process data in that range.

The following figure shows the relationship between stream data and the window.

Figure 2-6: Relationship between stream data and the window



The stream data (tuples) in the range defined by the window shown in this figure are temporarily stored in memory for processing.

A window defines the range of the stream data elements being processed, which can be defined in terms such as time, number of tuples, and so on. For details about specifying windows, see *2.2.2 Using data manipulation CQL to specify operations on stream data*.

2.2 Using CQL to process stream data

Stream data is processed according to the instructions in the query definition file used by the system. The query definition file uses CQL to describe the stream data type (`STREAM`) and the queries. These CQL instructions are called *CQL statements*.

There are two types of CQL statements used for writing query definition files:

- Definition CQL

  These CQL statements are used to define streams and queries.

- Data manipulation CQL

  These CQL statements are used to process the stream data.

This section describes how to use definition CQL to define streams and queries, and how to use data manipulation CQL to perform processing on stream data.

For additional details about CQL, see the *uCosminexus Stream Data Platform - Application Framework Application Development Guide*.

CQL statements consist of keywords, which have preassigned meanings, and items that you specify following a keyword. An item you specify, combined with one or more keywords, is called a *clause*. The code fragments discussed on the following pages are all clauses. For example, `REGISTER STREAM` *stream-name*, consisting of the keywords `REGISTER STREAM` and the user-specified item *stream-name*, is known as a `REGISTER STREAM` clause.

2.2.1 Using definition CQL to define streams and queries

CQL statements that are used to define streams and queries are called *definition CQL*. There are two types of definition CQL.

- `REGISTER STREAM` clauses
- `REGISTER QUERY` clauses

The following subsections explain how to specify each of these clauses.

(1) Defining a stream (REGISTER STREAM clause)

To define a stream, you use the definition CQL `REGISTER STREAM` clause.

In the `REGISTER STREAM` clause, you specify the stream name (the name of the input stream) and the schema specification character string (the content of the stream data that identifies it as the input stream). The following shows the format for specifying a `REGISTER STREAM` clause:

```
REGISTER STREAM stream-name
(schema-specification-character-string) ;
```

For example, the following shows a CQL statement that defines a stream for a temperature analysis system:

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
```

For the stream name, `temperature_stream` is specified. The parameters `observation_time` (observation time), `id` (observation site ID), and `temperature` (measured temperature) are specified using the `TIME`, `INTEGER`, and `INTEGER` data types, respectively.

(2) Defining a query (REGISTER QUERY clause)

To define a query, you use the definition CQL `REGISTER QUERY` clause. In the `REGISTER QUERY` clause, you specify the name of the query, a stream clause, a `SELECT` clause, a `FROM` clause, and a `WHERE` clause, in that order. For the query name, you specify the name of the stream that is to be output after the stream data has been processed by the query (output stream).

The following table shows what the stream clause, `SELECT` clause, `FROM` clause, and `WHERE` clause, which make up the CQL `REGISTER QUERY` clause, do.

Table 2-1: Correspondence between query actions and clauses in a CQL statement

Clause in a CQL statement	Query action
Stream clause	Specifies a stream operation action.
`SELECT` and `WHERE` clauses	Specifies a relation operation action.
`FROM` clause	Specifies a window operation action.

The format for specifying a `REGISTER QUERY` clause is shown below:

```
REGISTER QUERY query-name
stream-clause (
SELECT-clause
FROM-clause
WHERE-clause);
```

The following paragraphs explain the clauses in the `REGISTER QUERY` clause.

Stream clause

In the stream clause, you specify how to output the stream data. Depending on exactly what you want to do, you can specify an `ISTREAM` clause, a `DSTREAM` clause, or an `RSTREAM` clause. For details about these clauses, see *2.2.2(3) Stream operations (outputting the data processing results)*.

`SELECT` clause

In the `SELECT` clause, you specify the specific data to be extracted from the tuples that are to be processed. To identify the data you want, specify the data item names used in the schema specification character string of the `REGISTER STREAM` clause. The actual tuples to be processed are determined by the `WHERE` clause.

`FROM` clause

In the `FROM` clause, you specify the name of the stream that the query is to process, and the processing window. Depending on exactly what you want to do, you specify one of these four window types: a `ROWS` window, a `RANGE` window, a `NOW` window, or a `PARTITION BY` window.

You specify the stream data using the stream name specified in the `REGISTER STREAM` clause. Following the stream name, you specify the window type and size enclosed in square brackets (`[ ]`). For details about windows, see *2.2.2(1) Window operations (retrieving data for analysis)*.

For a query to process a relation, you must specify the name of the relation.

`WHERE` clause

In the `WHERE` clause, you specify the criteria for selecting which tuples retrieved from the window are to be processed.

For example, a query following the code for the temperature analysis system described above in *(1) Defining a stream (REGISTER STREAM clause)* is shown below.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY sensor_filter
ISTREAM (
SELECT id, temperature
FROM temperature_stream[ROWS 3]
WHERE id = 1);
```

The following paragraphs explain the items specified in the above `REGISTER QUERY` clause.

- Query name

  The query name is specified as `sensor_filter`. This is the name of the input stream used to acquire the stream data to be processed.

- Stream clause

  The operator `ISTREAM` is specified as the stream operation.

- `SELECT` clause

  The data items `id` and `temperature` are specified as the information to be retrieved from the tuples selected by the `WHERE` clause. These are data item names

specified in the schema specification character string of the `REGISTER STREAM` clause.

- `FROM` clause

  The line `temperature_stream[ROWS 3]` specifies the name of the stream and of the window to be processed by the query. The stream name is the one specified in the `REGISTER STREAM` clause. For the window, `[ROWS 3]` is specified. This means that three most recent tuples are to be retrieved from the stream data in descending time order.

- `WHERE` clause

  Here, the condition `id = 1` specifies which tuples are to be operated on from those that are retrieved in the window. This means that tuples with an ID equal to `1` are to be selected from the tuples retrieved in the window.

You can also specify other clauses in the `REGISTER QUERY` clause that are not described here. For details on these other clauses, see the *uCosminexus Stream Data Platform - Application Framework Application Development Guide*.

2.2.2 Using data manipulation CQL to specify operations on stream data

There are three types of data manipulation CQL operations:

- Window operations
- Relation operations
- Stream operations

The descriptions in the following subsections are based on the example of the temperature analysis system described in *2.2.1(1) Defining a stream (REGISTER STREAM clause)*.

(1) Window operations (retrieving data for analysis)

A *window operation* is used to retrieve data for analysis from stream data. In the query, a window is specified in the `FROM` clause following the stream name. There are four types of window operations:

- `ROWS` window
- `RANGE` window
- `NOW` window
- `PARTITION BY` window

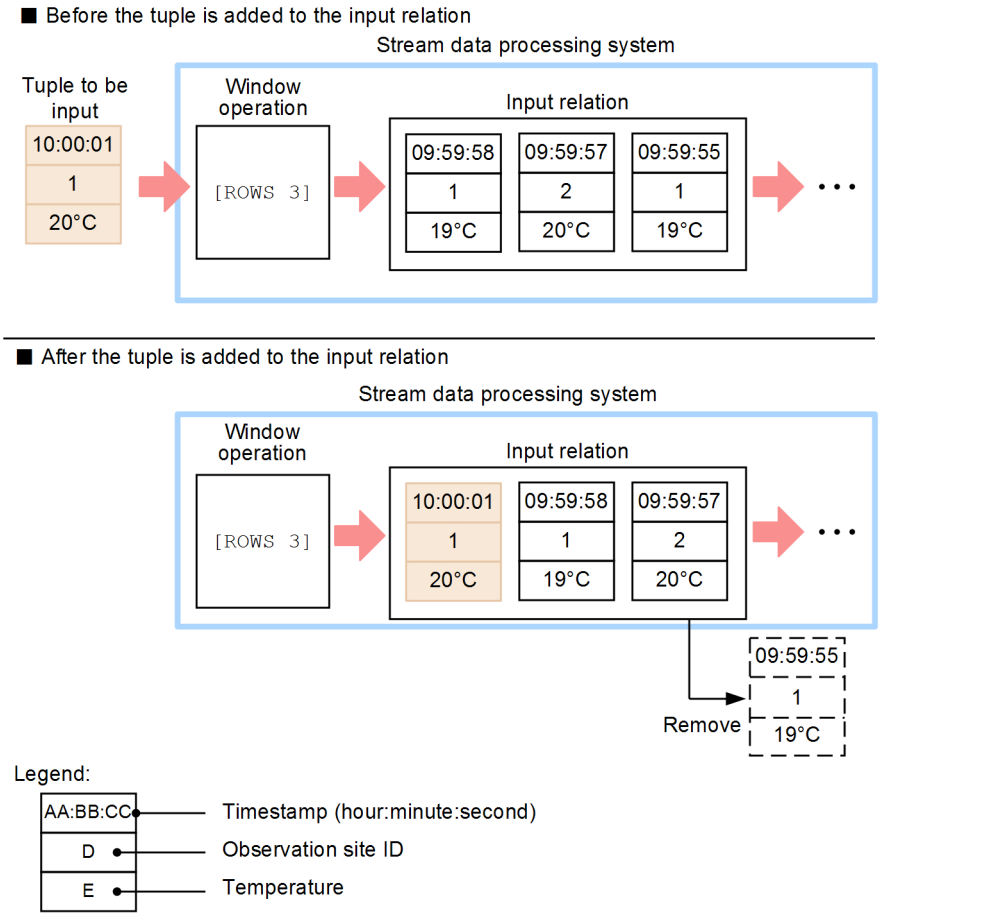
The following subsections explain these window operations.

ROWS window

A ROWS window uses a count to specify the number of tuples to retrieve from the stream data. The input relation generated by a ROWS window is a tuple group beginning with the most recent tuple and going back a specified number of tuples. In a ROWS window, new tuples are added to the beginning of the input relation with each new tuple that is received. Similarly, tuples that exceed the tuple count are removed from the end of the input relation.

In a ROWS window, you specify the number of tuples to retrieve. For example, if you specify [ROWS 3], the input relation consists of the three most recent tuples that have been retrieved in order starting with the most recent one. The following figure shows a ROWS window being used to retrieve data for analysis.

Figure 2-7: Using a ROWS window to retrieve data for analysis



In the window operation shown in this figure, three tuples are to be retrieved,

which means that the input relation always maintains exactly three tuples. To ensure this, when a new tuple is added to the input relation, the oldest tuple in the input relation is removed.

`RANGE` window

A `RANGE` window uses a unit of time to specify the tuples to retrieve from the stream data. The input relation generated by the `RANGE` window is a tuple group beginning with the most recent tuple and going back a specified period of time.

In a `RANGE` window, you specify the time period in which to retrieve tuples. For example, if you specify `[RANGE 3 SECOND]`, the input relation consists of all tuples that have been retrieved that have a timestamp within three seconds of the most recent tuple.

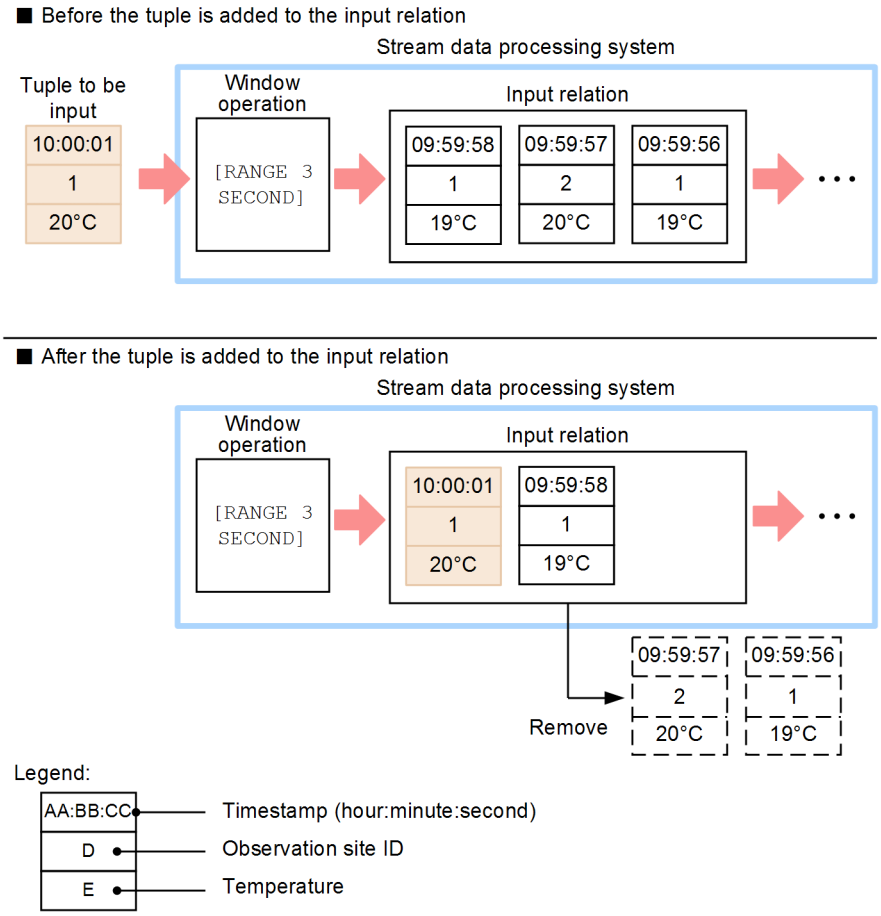
The following table lists the units that can be used for specifying a time period.

Table 2-2: Time units that can be specified in a CQL statement

Specification in CQL statement	Unit
`MILLISECOND`	Millisecond
`SECOND`	Second
`MINUTE`	Minute
`HOUR`	Hour
`DAY`	Day

The following figure shows a `RANGE` window being used to retrieve data for analysis.

Figure 2-8: Using a RANGE window to retrieve data for analysis



In the window operation shown in this figure, all tuples whose timestamp is within three seconds of the most recent tuple (10:00:01 to 09:59:58) will be retrieved. This means that any tuples that no longer satisfy this condition when a new tuple is added to the input relation will be removed from the input relation.

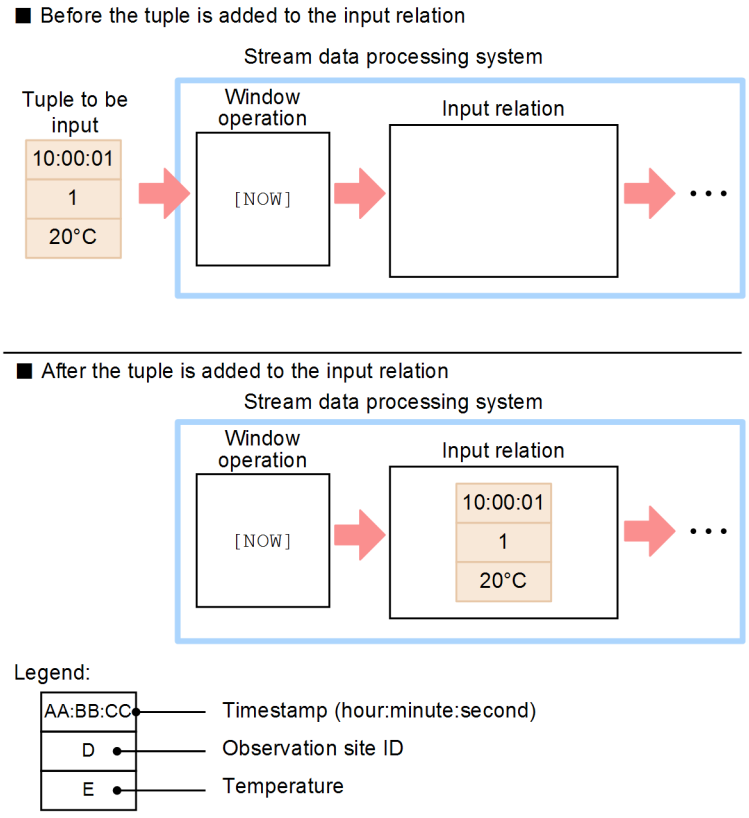
When you use a `RANGE` window, depending on the input data, the number of tuples handled by Stream Data Platform - AF could become quite large and the amount of memory required may increase proportionately. In this case, you can use the time division function to prevent the amount of memory required from increasing too much. For details about the time division function, see the *uCosminexus Stream Data Platform - Application Framework Application Development Guide*.

NOW window

A NOW window specifies that tuples are only to be processed at the time they arrive. If multiple tuples with the same timestamp arrive simultaneously, all of them are processed together. Because any tuple is removed from the NOW window as soon as it is processed, only the tuple (or tuples) with the most recent timestamp are present in the input relation generated by a NOW window.

You use the [NOW] specification to specify a NOW window. The following figure shows a NOW window being used to retrieve data for analysis.

Figure 2-9: Using a NOW window to retrieve data for analysis



In the window operation shown in this figure, only the tuple that arrives at that particular point in time is selected for processing. Because any tuple in the input relation is removed after being processed, no tuples are present in the input relation when a new tuple is added.

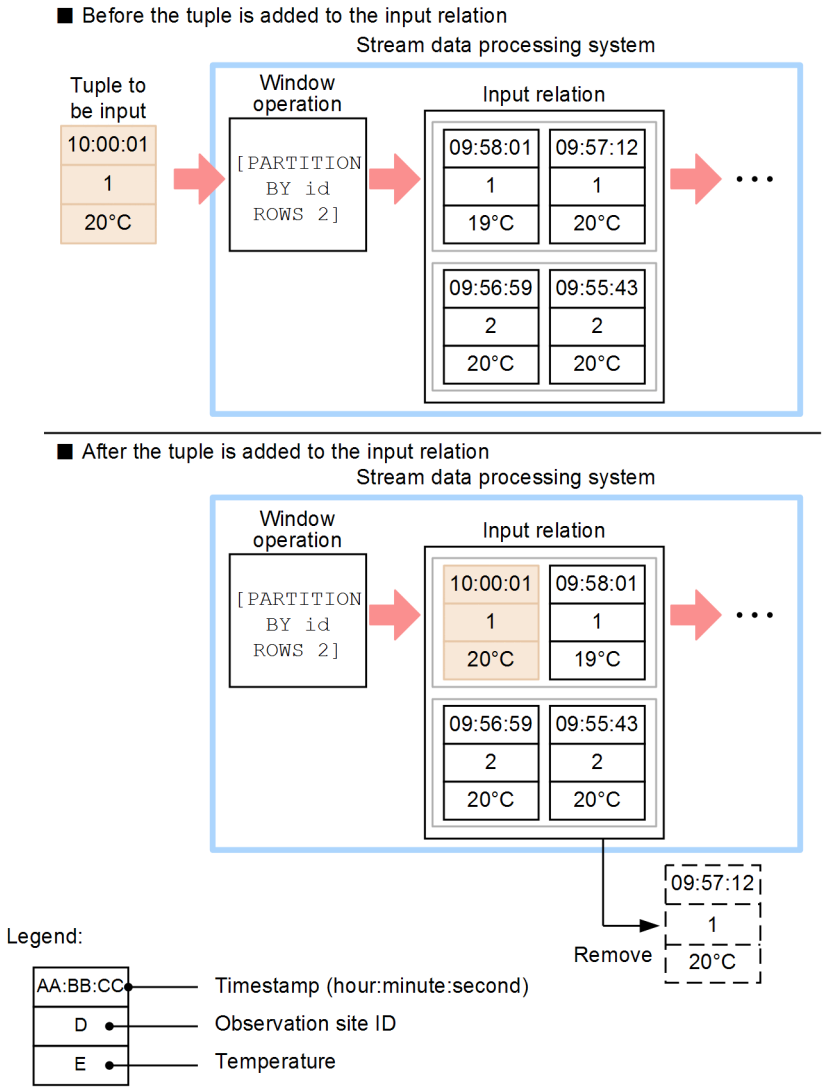
PARTITION BY window

A PARTITION BY window uses data values to specify the tuples to retrieve from

the stream data. This window is used together with a `ROWS` window. The input relation generated by a `PARTITION BY` window is a tuple group beginning with the most recent tuple containing the specified data item and going back a specified number of tuples.

In a `PARTITION BY` window, you enter the data item names to use for selecting. You follow these with `ROWS`, in which you specify the number of tuples to retrieve for each group. For example, if you specify `[PARTITION BY id ROWS 2]`, for each ID the input relation consists of the two most recent tuples that have been retrieved in the order in which they were received. The following figure shows a `PARTITION BY` window being used to retrieve data for analysis.

Figure 2-10: Using a PARTITION BY window to retrieve data for analysis



In the window operation shown in this figure, two tuples are to be retrieved for each ID that can be specified in the tuples. This means that, when a new tuple is added to the input relation, the oldest tuple with the same ID as the new tuple is removed from the input relation.

(2) Relation operations (processing the retrieved data)

A *relation operation* is used to process the data retrieved by the window operation. The following operations are available:

- Retrieve data that satisfies a condition
- Perform data calculations
- Summarize data
- Categorize and then summarize data
- Join data streams
- Link queries

These operations can be specified in the `SELECT` and `WHERE` clauses using arithmetic operators, comparison operators, logical operators, and aggregate functions.

The following tables list the comparison operators and aggregate functions that can be specified.

Table 2-3: Comparison operators that can be used in CQL statements

Comparison operator	Usage example	Meaning of usage example
`<=`	`A <= B`	A is less than or equal to B
`>=`	`A >= B`	A is greater than or equal to B
`<`	`A < B`	A is less than B
`>`	`A > B`	A is greater than B
`=`	`A = B`	A is equal to B
`!=`	`A != B`	A is not equal to B

Table 2-4: Aggregate functions that can be used in CQL statements

Function	Description
`AVG`	Computes the average of all values.
`COUNT`	Counts the number of items.
`MAX`	Determines the maximum value.
`MIN`	Determines the minimum value.
`SUM`	Computes the sum of all values.

The logical operators that can be specified in CQL statements vary depending on the clause. For details about the logical operators that can be specified in CQL statements, see the *uCosminexus Stream Data Platform - Application Framework Application Development Guide*.

For examples of how to implement relation operations, see *2.3 Implementation examples of using CQL to process stream data*.

(3) Stream operations (outputting the data processing results)

A *stream operation* is used to take the results of a relation operation, convert it to stream data and output it. Stream operations are specified in the stream clause directly following the `REGISTER QUERY` clause. The following three types of stream operations are available:

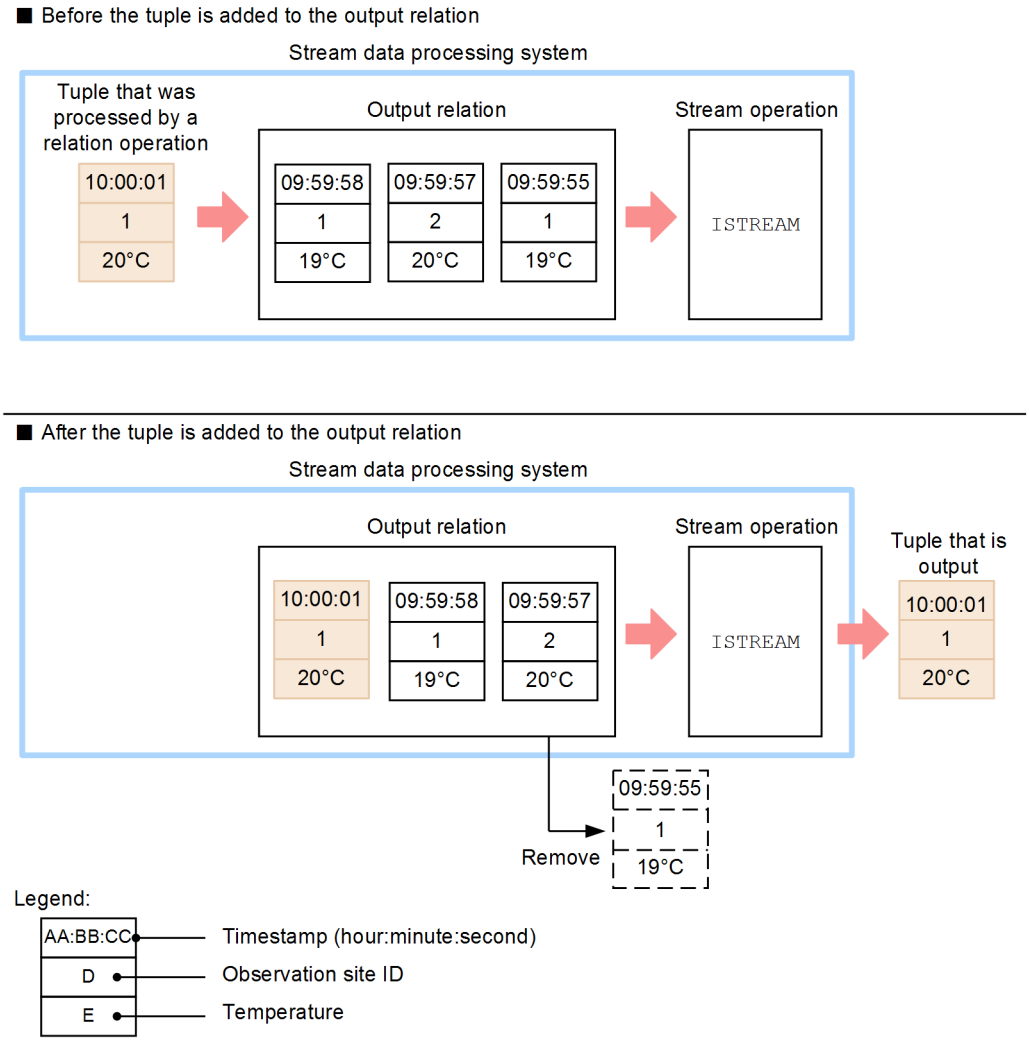
- `ISTREAM`
- `DSTREAM`
- `RSTREAM`

Assuming that the window operation `[ROWS 3]` is specified, the following sections explain each of these stream operations.

`ISTREAM`

An `ISTREAM` stream operation outputs the tuples that were added to the output relation. Each time the output relation changes, `ISTREAM` compares the output relation before and after the change, and only outputs the tuples that were most recently added. The following figure shows the processing results that are output by `ISTREAM`.

Figure 2-11: Processing results that are output by ISTREAM



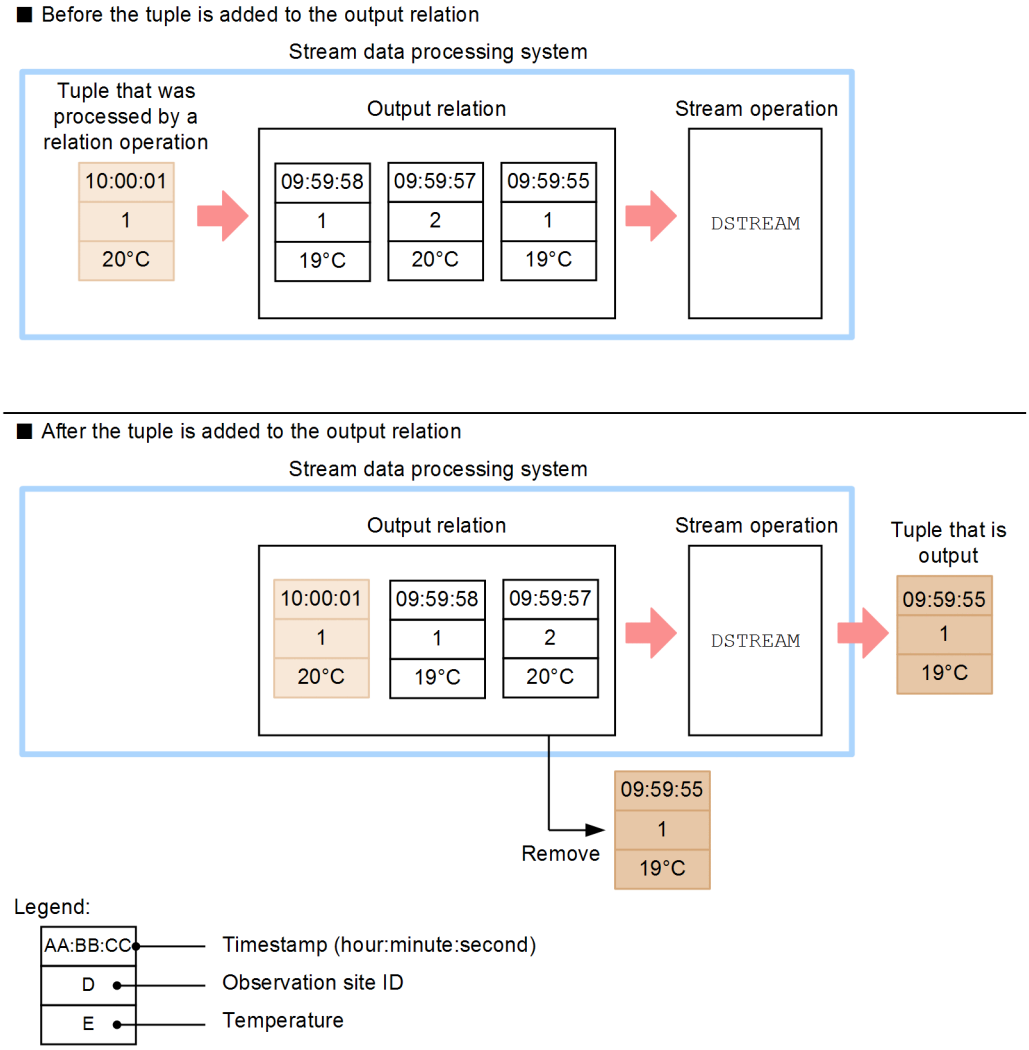
When a tuple processed by a relation operation is added to the output relation, the oldest tuple in the output relation is removed. Because `ISTREAM` is specified as the stream operation, when this occurs, only the tuple that was added to the output relation is output.

`DSTREAM`

A `DSTREAM` stream operation outputs the tuples that were removed from the output relation. Each time the output relation changes, `DSTREAM` compares the output relation before and after the change, and outputs the tuples that were

removed. The following figure shows the processing results that are output by DSTREAM.

Figure 2-12: Processing results that are output by DSTREAM

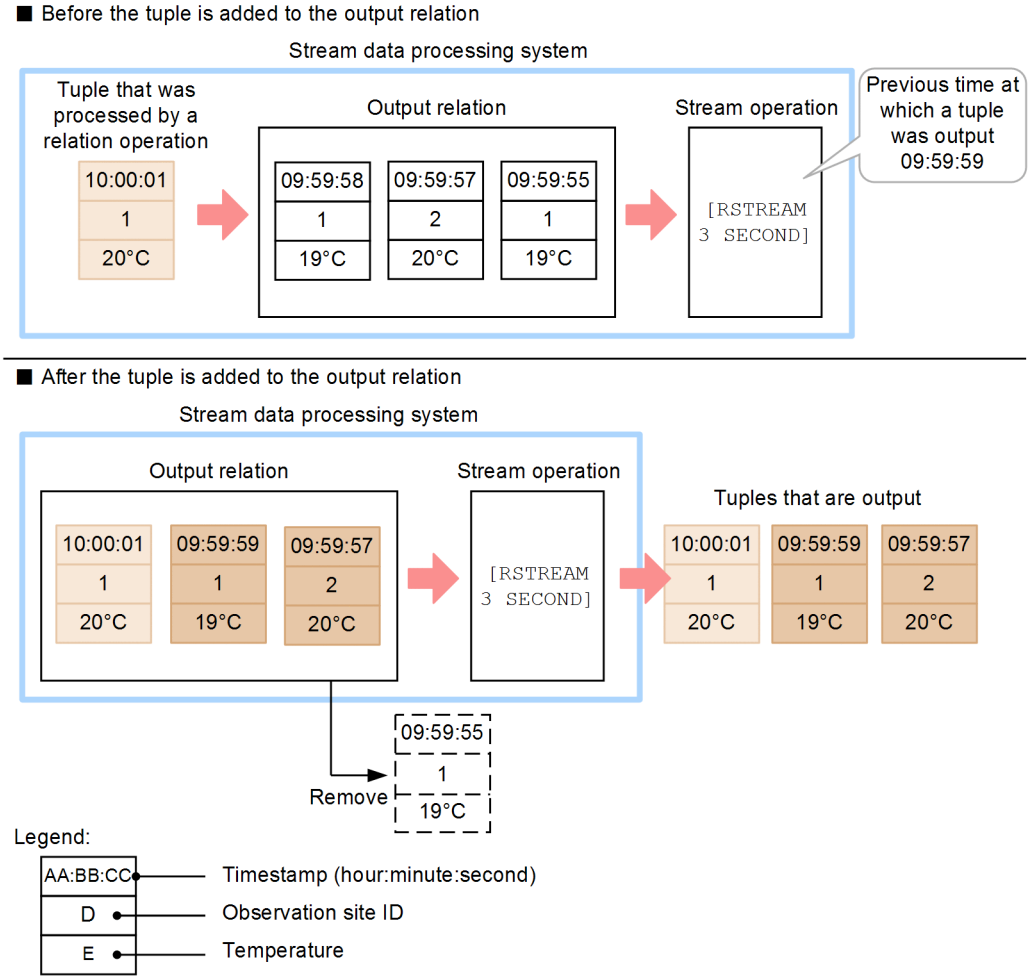


When a tuple processed by a relation operation is added to the output relation, the oldest tuple in the output relation is removed. Because DSTREAM is specified as the stream operation, when this occurs, only the tuple that was removed from the output relation is output.

RSTREAM

An RSTREAM stream operation outputs all tuples in the output relation at specific time intervals. When you specify an RSTREAM clause, you enclose the time interval for outputting the stream in square brackets ([ ]). For example, you could specify [RSTREAM 1 MINUTE] or [RSTREAM 3 SECOND]. You can use the units listed in *Table 2-2 Time units that can be specified in a CQL statement* to specify the time interval. The following figure shows the processing results that are output by RSTREAM.

Figure 2-13: Processing results that are output by RSTREAM



RSTREAM 3 SECOND is specified as the stream operation, so every three seconds (as determined by the system time) all tuples in the output relation are output.

2.3 Implementation examples of using CQL to process stream data

The implementation examples explained in this section are based on the information covered in *2.2 Using CQL to process stream data*.

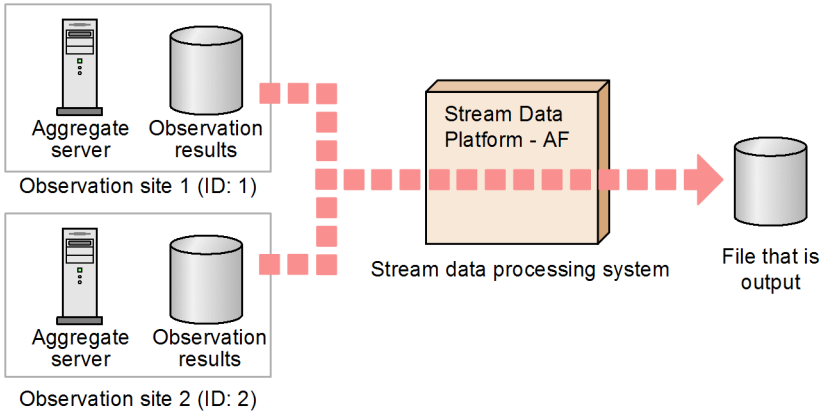
Using the temperature analysis system as an example, this section explains how query statements are used to process stream data. The following operations are demonstrated in these examples:

- Retrieve data that satisfies a condition
- Perform data calculations
- Summarize data
- Categorize and then summarize data
- Join data streams
- Link queries

2.3.1 Assumed system

The following figure shows the system assumed for these implementation examples.

Figure 2-14: Temperature analysis system assumed for the stream data processing implementation examples



In this temperature analysis system, observation sites 1 (ID: 1) and 2 (ID: 2) each monitor temperatures, and each send their observation results (in a file) to a sequential stream data processing system. The stream data processing system summarizes and analyzes the data that is sent by the observation sites as instructed by the queries, and then outputs the results to a file.

2.3.2 Retrieving data that satisfies a condition

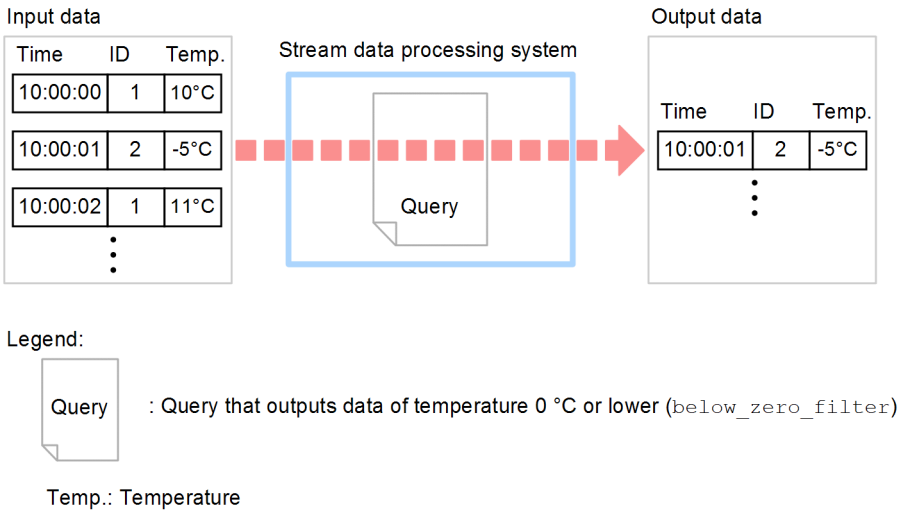
This section uses the following two queries as examples of retrieving data that satisfies a condition.

- Query that outputs data of temperature 0 °C or lower
- Query that outputs data of temperature between -10 °C and 0 °C (inclusive).

(1) Query that outputs data of temperature 0 °C or lower

This subsection explains a query designed to output data that satisfies a single condition: output data of temperature 0 °C or lower. The following figure shows the input and output data present when this query is executed.

Figure 2-15: Input and output data present when a query that satisfies a single condition is executed



Code

To output data that satisfies the condition of temperature 0 °C or lower, you use a comparison operator in the `WHERE` clause to specify a data selection condition.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY below_zero_filter
ISTREAM (
SELECT observation_time, id, temperature
FROM temperature_stream[ROWS 1]
WHERE temperature <= 0);
```

Explanation

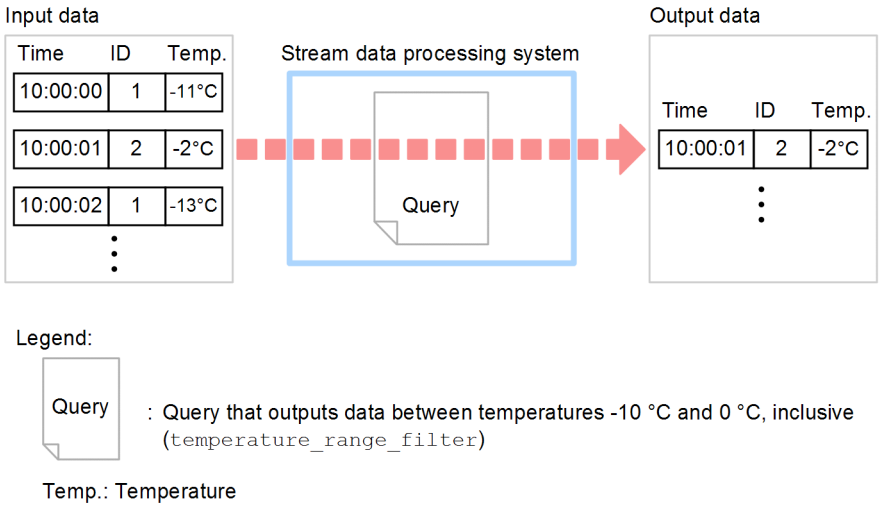
The processing target of this query is the single most recent tuple to meet the condition. The FROM clause specifies a ROWS window, which retrieves tuples from the stream data in terms of the number of tuples.

The output data is data of temperature 0 °C or lower. In the WHERE clause, temperature <= 0 is specified. This specification causes the clause to compare the value of temperature to 0, and selects tuples in which the value of temperature is less than or equal to 0.

(2) Query that outputs data of temperature between -10 °C and 0 °C (inclusive)

This subsection explains a query designed to output data that satisfies two conditions: output data of temperature between -10 °C or higher and 0 °C or lower (inclusive). The following figure shows the input and output data present when this query is executed.

Figure 2-16: Input and output data present when a query designed to output data that satisfies two conditions is executed



Code

To output data that satisfies the conditions of temperatures -10 °C or higher and 0 °C or lower (inclusive), you use comparison operators and the logical operator AND in the WHERE clause to specify a data selection condition.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY temperature_range_filter
ISTREAM (
SELECT observation_time, id, temperature
FROM temperature_stream[ROWS 1]
WHERE -10 <= temperature AND temperature <= 0);
```

Explanation

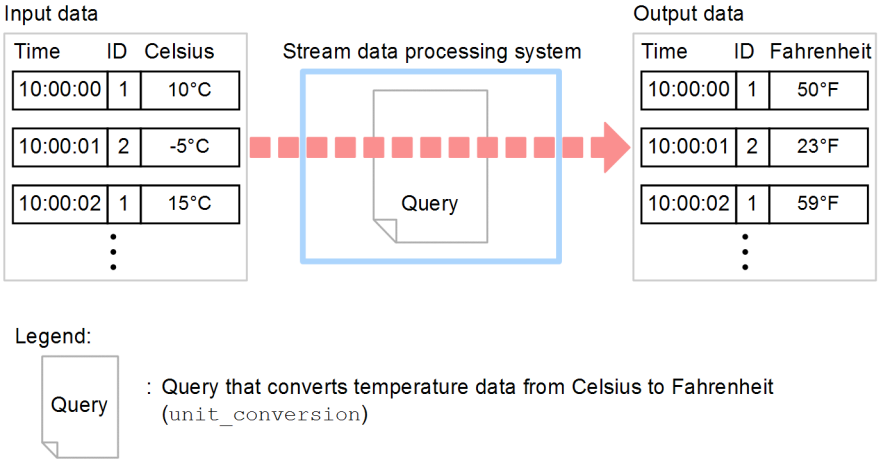
The processing target of this query is the single most recent tuple to meet the condition. The `FROM` clause specifies a `ROWS` window, which retrieves tuples from the stream data in terms of the number of tuples.

The output data is data of temperature -10 °C or higher and 0 °C or lower. In the `WHERE` clause, `-10 <= temperature` and `temperature <= 0` are specified as linked conditions. This specification causes the clause to compare the value of `temperature` to `-10`, selecting tuples in which the value of `temperature` is greater than or equal to `-10`, AND it compares the value of `temperature` to `0`, selecting tuples in which the value of `temperature` is less than or equal to `0`.

2.3.3 Performing data calculations

This subsection explains a query that performs data calculations, using an example that converts temperature data that was collected in degrees Celsius to degrees Fahrenheit. To convert Celsius to Fahrenheit, the query needs to include a mathematical formula. The following figure shows the input and output data present when this query is executed.

Figure 2-17: Input and output data present when a query designed to perform data calculations is executed



Code

To convert temperature from Celsius to Fahrenheit, you use arithmetic operations in the `SELECT` clause to express a mathematical formula.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY unit_conversion
ISTREAM (
SELECT observation_time, id,
temperature*9/5+32 AS fahrenheit_temperature
FROM temperature_stream[ROWS 1]);
```

Explanation

The processing target of this query is the single most recent query. The `FROM` clause specifies a `ROWS` window, which retrieves tuples from the stream data in terms of the number of tuples.

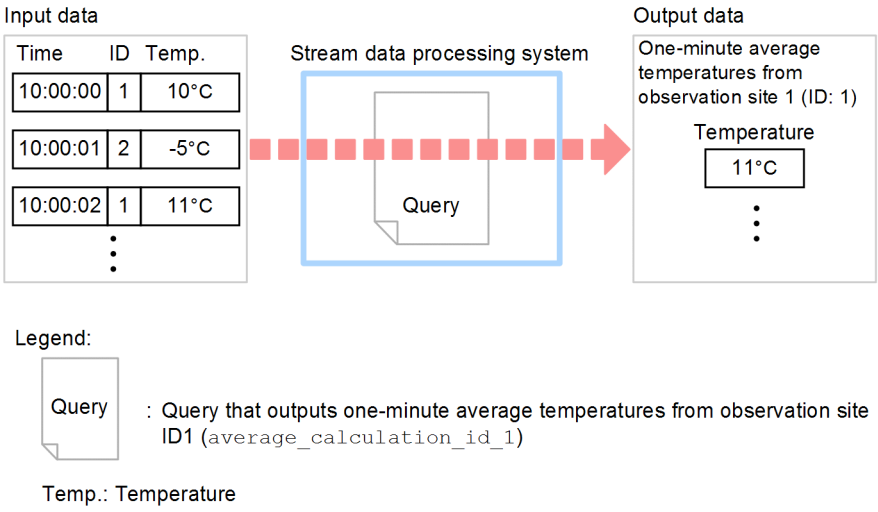
The output data is temperature data expressed in Fahrenheit. In the `SELECT` clause, the code fragment `temperature*9/5+32` is the arithmetic expression that converts from Celsius to Fahrenheit.

To output calculated data, a name must be assigned to the calculation results. The keyword `AS` is used to specify the name of the resulting data. In the CQL code, `fahrenheit_temperature` is specified as the name for the data that results from the conversion to Fahrenheit.

2.3.4 Summarizing data

This subsection explains a query that summarizes data, using an example that outputs average temperatures in one-minute intervals from observation site 1 (ID: 1). First, the query summarizes one minute of temperature data from ID1, and then it derives an average from the summary results. The following figure shows the input and output data present when this query is executed.

Figure 2-18: Input and output data present when a query designed to summarize data is executed



Code

To summarize data, you use an aggregate function in the SELECT clause to specify how you want the data to be summarized. To target specific data for summarization, you use a comparison operator in the WHERE clause to specify a data selection condition.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY average_calculation_id_1
ISTREAM (
SELECT AVG(temperature) AS average_temperature
FROM temperature_stream[RANGE 1 MINUTE]
WHERE id = 1);
```

Explanation

The processing target of this query is all of the tuples from the last minute. The FROM clause specifies a RANGE window, which retrieves tuples from the stream data in terms of time.

The output data is one-minute averages of the temperatures observed at site 1 (ID1). In the SELECT clause, the aggregate function AVG is specified to compute the average value of temperature, and average_temperature is specified as the name of the data item into which the results are output. In the WHERE clause, id = 1 is specified as the target of the computational processing.

When data processing is specified in both the SELECT and the WHERE clauses, the

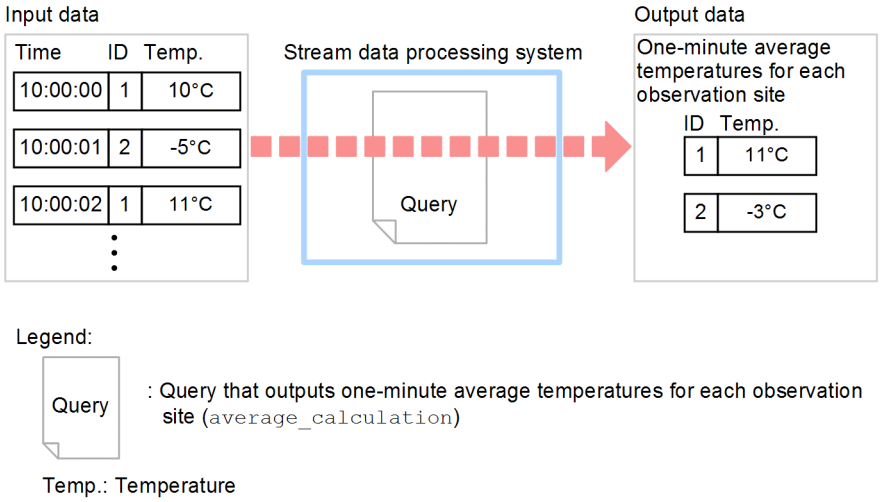
WHERE clause is executed first, and then the SELECT clause is executed on the results from the WHERE clause.

The tuple group selected by the WHERE clause, which is executed first, is called an *intermediate relation*. In this query, this is the tuple group from the last minute that is selected only from observation site ID1.

2.3.5 Categorizing and then summarizing data

This subsection explains a query that categorizes multiple data items and then summarizes the data, using an example that outputs one-minute average temperatures for each observation site. The following figure shows the input and output data present when this query is executed.

Figure 2-19: Input and output data present when a query designed to categorize and summarize data is executed



Code

To categorize data, you use the GROUP BY clause to specify how the data is to be categorized. You specify the GROUP BY clause immediately after the WHERE clause (or immediately following the FROM clause if the WHERE clause is not specified).

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY average_calculation
ISTREAM (
SELECT id, AVG(temperature) AS average_temperature
FROM temperature_stream[RANGE 1 MINUTE]
GROUP BY id);
```

Explanation

The processing target of this query is all of the tuples from the last minute. The `FROM` clause specifies a `RANGE` window, which retrieves tuples from the stream data in terms of time.

The output data is one-minute averages of the temperatures from each observation site. In the `SELECT` clause, the aggregate function `AVG` is specified to compute the average value of `temperature`, and `average_temperature` is specified as the name of the data item into which the results are output. In the `GROUP BY` clause, `id` is specified to group data by observation site.

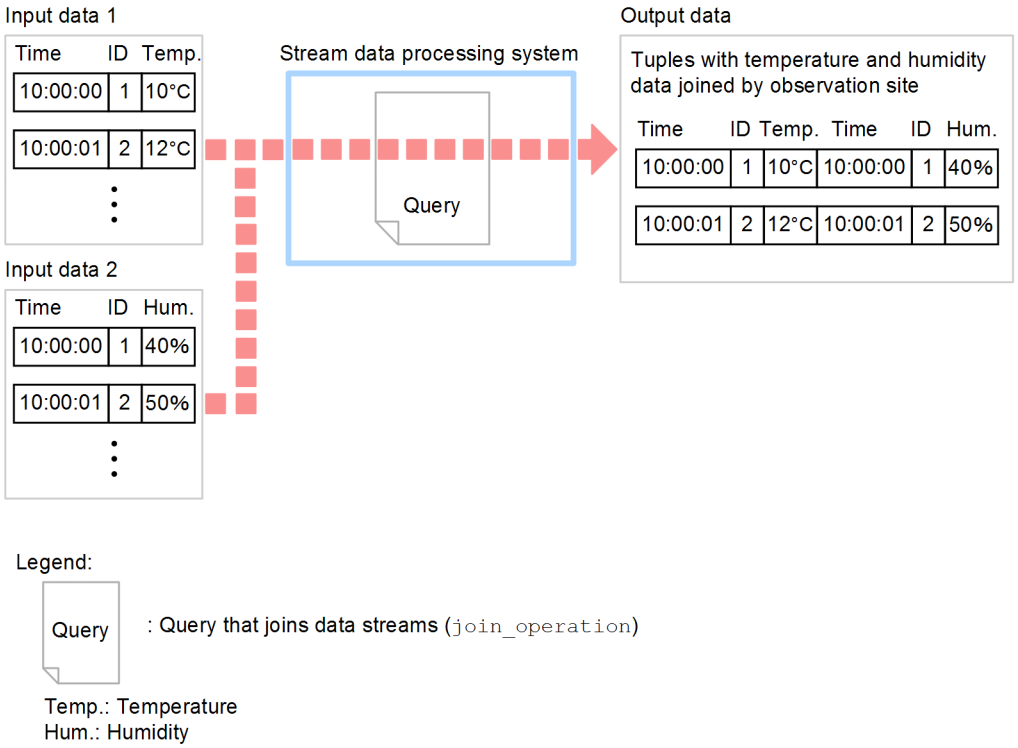
When data processing is specified in both the `SELECT` clause and the `GROUP BY` clause, the `SELECT` clause is executed according to the category defined in the `GROUP BY` clause.

2.3.6 Joining data streams

Selecting tuples from multiple data streams and performing computational processing to consolidate these tuples into a single tuple is called *joining data streams*.

This subsection explains a query that joins data streams, using an example that first joins two data streams (temperature and humidity), and then joins tuples from the same observation site. The following figure shows the input and output data present when this query is executed.

Figure 2-20: Input and output data present when a query that joins data streams is executed



Code

To accept input from multiple data streams, you specify a comma (,) in the `FROM` clause to delimit the data streams. In this case, a window operation must be specified for each data stream. To subsequently join the data, you specify a join condition in the `WHERE` clause.

In the following code, the name of the humidity stream is `humidity_stream`, and the name of the humidity data item is `humidity`.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER STREAM humidity_stream
(observation_time TIME, id INTEGER, humidity INTEGER);
REGISTER QUERY join_operation
ISTREAM (
SELECT temperature_stream.observation_time AS temperature_stream_time,
temperature_stream.id AS temperature_stream_id,
temperature_stream.temperature,
humidity_stream.observation_time AS humidity_stream_time,
humidity_stream.id AS humidity_stream_id,
humidity_stream.humidity
FROM temperature_stream[PARTITION BY id ROWS 1],
humidity_stream[PARTITION BY id ROWS 1]
WHERE temperature_stream.id = humidity_stream.id);
```

Explanation

The processing target of this query is the single most recent tuple from each observation site. In the CQL `ISTREAM` statement above, `PARTITION BY` windows are specified to retrieve the most recent single tuple from each observation site.

The output data joins two data streams, which contain temperature and humidity data, and then joins tuples by observation site. In the `WHERE` clause, `temperature_stream.id = humidity_stream.id` is specified as the condition that joins tuples of the same ID.

In the `SELECT` clause, output data names are specified for the data items in the tuples that are joined. When multiple data streams are input, different data streams might have data items with the same name. To distinguish which data stream such items belong to, a period (.) is added as a delimiter between the stream data name and the data item name. Then, to avoid having identically named data items, `AS` is used to assign unique data item names to the input data.

2.3.7 Linking queries

In some cases, you cannot achieve the desired results by executing a single query. If this is the case, you can send the processing results of the first query (first-stage) to the second query (second-stage). This is called *query linking*. The following two types of query linking are possible:

- Stream data-based query linking
- Relation-based query linking

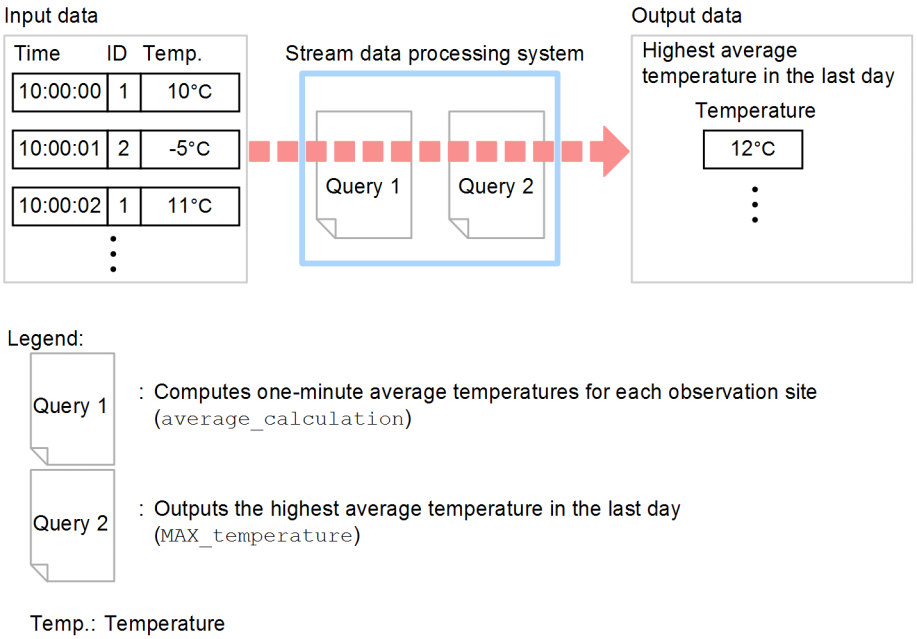
When you link queries, the data processed by the first-stage query might be different from the data processed by the second-stage query. For example, if the first-stage query analyzes data in one-minute intervals, and the second-stage query analyzes data in one-hour intervals, it is best to use stream data-based query linking. Conversely, if the data being analyzed by both the first-stage query and the second-stage query is the same, it is best to use relation-based query linking.

The following subsections explain each type of query linking.

(1) Stream data-based query linking

This subsection explains stream data-based query linking, using an example that computes one-minute average temperatures for each observation site, and then outputs the highest average temperature in the last day. The following figure shows the input and output data present when these queries are executed.

Figure 2-21: Input and output data present when stream data-linked queries are executed



The first-stage query (query 1) computes one-minute average temperatures for each observation site, and the second-stage query (query 2) computes the highest average temperature in the last day.

Code

To link queries, you specify the name of the first-stage query in the `FROM` clause of the second-stage query.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY average_calculation
ISTREAM (
SELECT id, AVG(temperature) AS average_temperature
FROM temperature_stream[RANGE 1 MINUTE]
GROUP BY id);
REGISTER QUERY MAX_temperature
ISTREAM (
SELECT MAX(average_temperature)
FROM average_calculation[RANGE 1 DAY]);
```

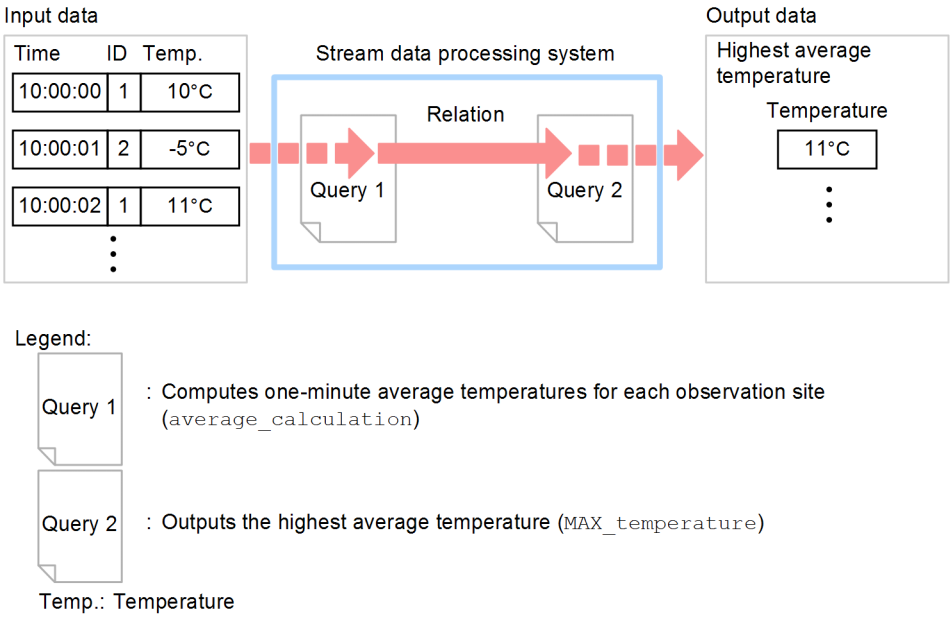
Explanation

The first-stage query (`average_calculation`) is the same as the query described in *2.3.5 Categorizing and then summarizing data*. The second-stage query (`MAX_temperature`) takes the stream data from the first-stage query and uses a window operation to retrieve tuples for analysis. The data analyzed by the second-stage query are all tuples from the last day. To achieve this, `[RANGE 1 DAY]` is specified in the `FROM` clause of the second-stage query.

(2) Relation-based query linking

This subsection explains relation-based query linking, using an example that computes the average temperature for each observation site, and then outputs the highest average temperature (maximum average temperature) at the current time. The following figure shows the input and the output data present when these queries are executed.

Figure 2-22: Input and output data present when relation-linked queries are executed



The first-stage query (query 1) computes one-minute average temperatures for each observation site, and the second-stage query (query 2) computes the highest average temperature.

Code

With relation-based query linking, you do not use a stream operation in the first-stage query, nor do you use a window operation in the second-stage query. You specify the name of the first-stage query in the `FROM` clause of the second-stage query.

Note that, if you do not use a stream clause, you do not need to enclose the `SELECT` clause in parentheses.

```
REGISTER STREAM temperature_stream
(observation_time TIME, id INTEGER, temperature INTEGER);
REGISTER QUERY average_calculation
SELECT id, AVG(temperature) AS average_temperature
FROM temperature_stream[RANGE 1 MINUTE]
GROUP BY id;
REGISTER QUERY MAX_temperature
ISTREAM (
SELECT MAX(average_temperature) AS MAX_temperature
FROM average_calculation);
```

Explanation

In relation-based query linking, the content of the output relation (average temperature for each observation site) of the first-stage query (`average_calculation`) is the same as that of the input relation (average temperature for each observation site) for the second-stage query (`MAX_temperature`).

Chapter

3. Exchanging Data

This chapter describes how data is passed from the input source to the stream data processing engine and then from the stream data processing engine to the output destination. Adaptors, which are the components used to exchange I/O data with the stream data processing engine, are explained based on the standard adaptors provided by Stream Data Platform - AF.

3.1 Types of adaptors used for exchanging data

With Stream Data Platform - AF, you use *adaptors* to pass data between the input source, the output destination, and the stream data processing engine. With adaptors, you can perform such actions as converting data formats and filtering data.

To use an adaptor, you connect it to the stream data processing engine. The following two methods can be used to connect an adaptor to the stream data processing engine:

- In-process connection

  A connection method used when the adaptor and the stream data processing engine run in the same process.

- RMI connection

  A connection method used when the adaptor and the stream data processing engine run in different processes. This connection method uses the Java RMI interface to connect the adaptor and the stream data processing engine.

For details about these connection methods, see the *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*.

There are also two types of adaptors: *standard adaptors*, provided with Stream Data Platform - AF, and *custom adaptors*, that you create by programming them in Java.

For both types, an adaptor that is used to pass data from the input data to the stream data processing engine is called an *input adaptor*. An adaptor that is used to pass data from the stream data processing engine to the output data is called an *output adaptor*.

3.1.1 Standard adaptors

The adaptors provided by Stream Data Platform - AF are called the *standard adaptors*. The actions that a standard adaptor performs in order to correctly pass data are defined in an adaptor definition file. For details about the adaptor definition file, see the *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*.

Using a standard adaptor, you can input the following types of data into the stream data processing engine:

- Files
- HTTP packets

A standard adaptor can also output the results of the stream data processing engine to the following destinations:

- Files

- The dashboard

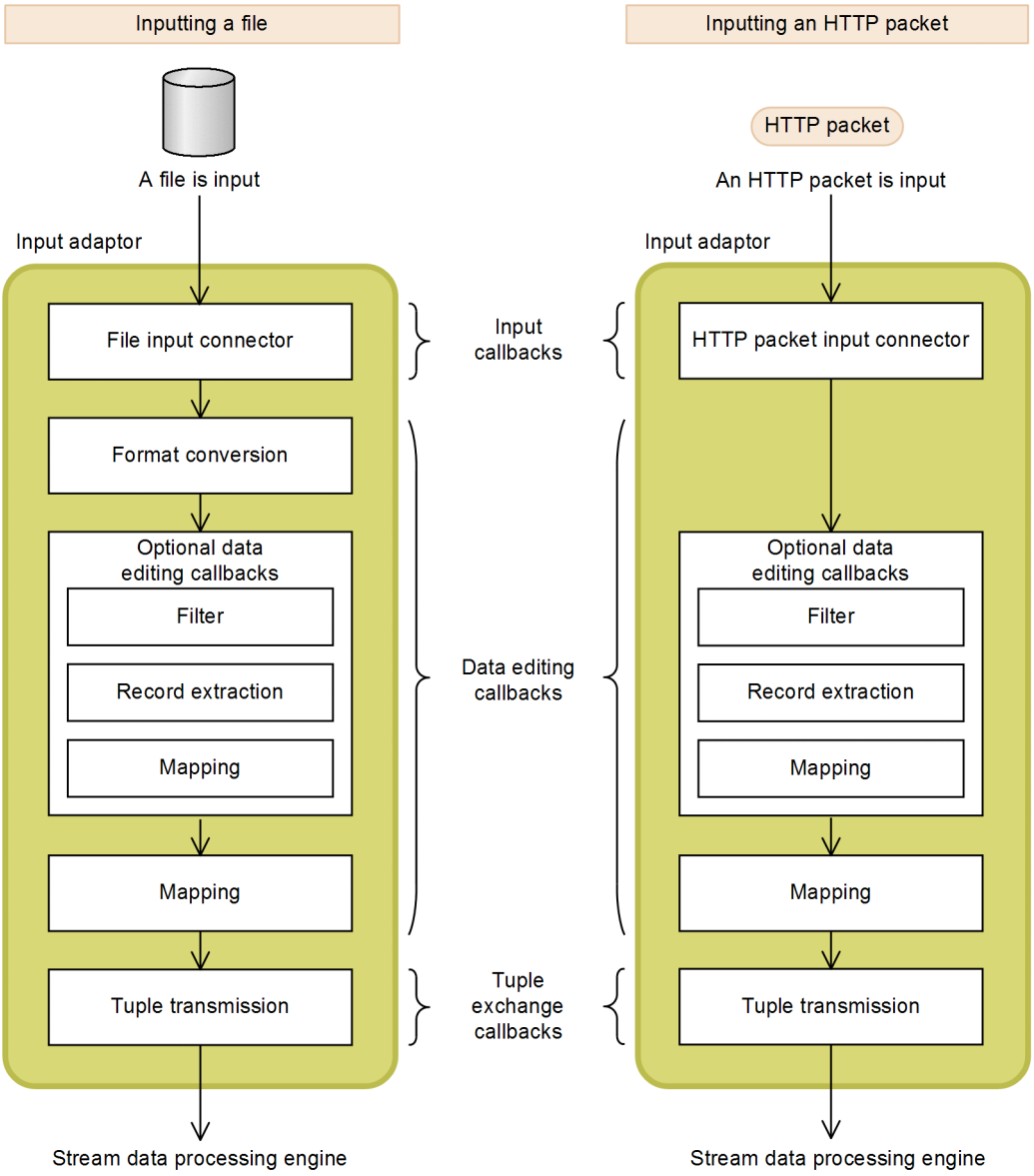
To input data from a data source other than the above, or to output results to an output destination other than the above, you need to use a custom adaptor. For details about custom adaptors, see *3.1.2 Custom adaptors*.

The standard adaptors function using units of processing called *callbacks*. Functions in the adaptor definition file are defined in terms of these callback units.

The standard adaptor callbacks are executed on input data one line (or row) at a time (for HTTP packets, they are executed by request message or by response message). This single line or message unit is called a *record*.

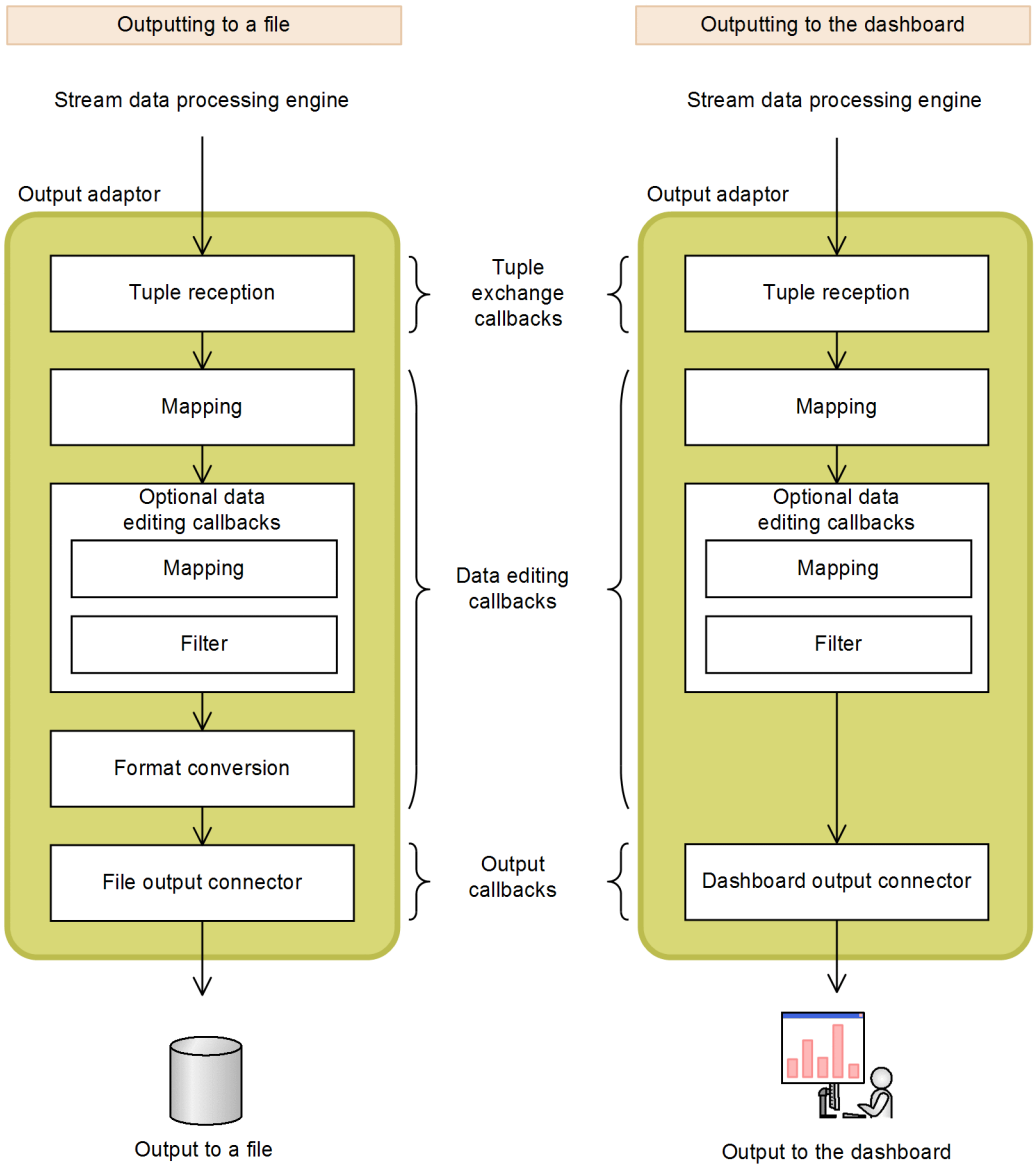
The following figure shows the organization of the callbacks in an input adaptor.

Figure 3-1: Organization of the callbacks in an input adaptor



The following figure shows the organization of the callbacks in an output adaptor.

Figure 3-2: Organization of the callbacks in an output adaptor



The following table lists and describes the functions of the standard adaptors.

Table 3-1: Functions of the standard adaptors

No.	Callback	Type	Function	Description
1	File input connector	Input callback	Inputs a file	Used for inputting a file.
2	HTTP packet input connector		Inputs HTTP packets	Used for inputting HTTP packets over a network.
3	Format conversion	Data editing callback	Converts the format of records during file input or output	Used only when the file input connector or file output connector is specified. There are two types of format conversion callbacks: one for the input adaptor and one for the output adaptor. For the input adaptor: This callback converts the records acquired from the input data (*input records*) to a format that can be processed by the stream data processing engine (*common records*). For the output adaptor: This callback converts the common records processed by the stream data processing engine to *output records*.
4	Filter		Filters records	Used to selectively extract records. When you use a filter callback, the adaptor can perform processing like that executed in the CQL code described in *2.3.2 Retrieving data that satisfies a condition*. To improve performance, we recommend that you use the adaptor to perform this kind of processing.
5	Record extraction		Extracts records	Used only with the input adaptor. This callback allows you to extract only the selected records that will then be joined to generate new records. When you use a record extraction callback, the adaptor can perform processing like that executed in the CQL code described in *2.3.6 Joining data streams*. To improve performance, we recommend that you use the adaptor to perform this kind of processing.

No.	Callback	Type	Function	Description
6	Mapping		Maps records	There are two types of record mapping callbacks: one for the input adaptor and one for the output adaptor. For the input adaptor: This callback converts common records to common records that are consistent with the format of the input stream. Also, if necessary, this callback will first convert the common records to common records that are suitable for processing by the next callback. For the output adaptor: This callback converts common records that are consistent with the format of the output stream to common records that are suitable for processing by the next callback. This callback can also further convert these common records to common records that are suitable for processing by the next callback.
7	Tuple transmission	Tuple exchange callback	Sends tuples to the stream data processing engine	Used by the input adaptor to send tuples to the stream data processing engine.
8	Tuple reception		Receives tuples processed by the stream data processing engine	Used by the output adaptor to receive tuples processed by the stream data processing engine.
9	File output connector	Output callback	Outputs a file	Used to output the results of the stream data processing engine to a file.
10	Dashboard output connector		Outputs to the dashboard	Used to output and display the results of the stream data processing engine to the dashboard.

For more details about callbacks, see the *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*.

Beginning in section *3.2 Inputting files*, this manual provides an overview of the following callbacks:

- File input connector
- HTTP packet input connector
- Filter

- Record extraction
- File output connector
- Dashboard output connector

The callbacks listed in the table are used in combination with one another. The following example describes a callback configuration.

Example of a callback configuration

Assume that you are monitoring the temperature at a number of observation sites, and that you want to summarize and analyze the observation results from one particular observation site, and then output those processing results to a file. To do this, you need to code callbacks that perform the following actions:

- Summarize and analyze only the data sent from observation site 1 (ID: 1).
- Output the processing results to a file.

The following two figures show the positioning of the adaptor callbacks used to implement this processing for the input adaptor and the output adaptor, respectively.

Figure 3-3: Example of positioning and processing of the input adaptor callbacks

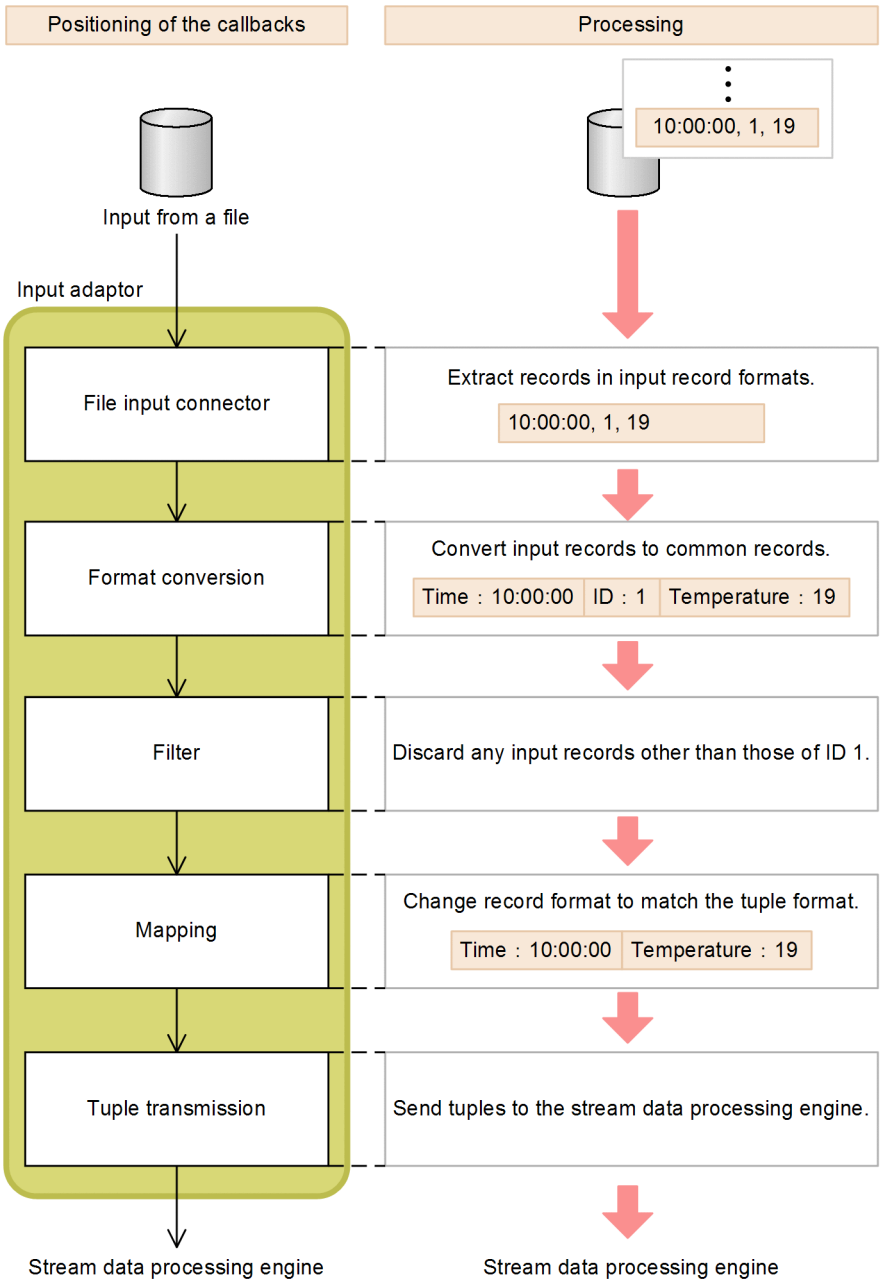
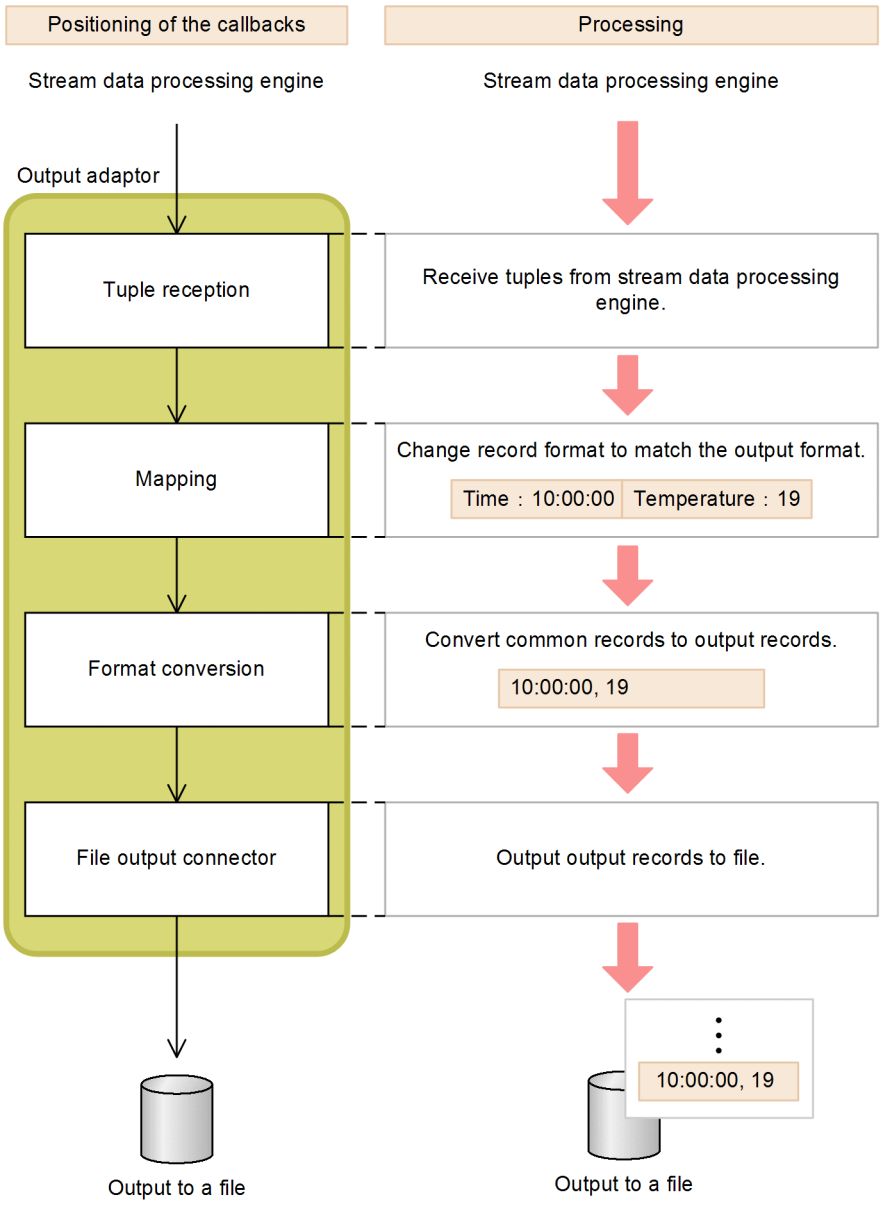


Figure 3-4: Example of positioning and processing of the output adaptor callbacks



The input adaptor uses a filter so only data from observation site 1 (ID: 1) is passed through for summary analysis, and the output adaptor uses a file output connector to send the processing results to a file.

3.1.2 Custom adaptors

Using the Java APIs provided with Stream Data Platform - AF, you can write Java applications to use as *custom adaptors*. In this way, you can build adaptors that perform a wide range of functions.

To create a custom adaptor, you need to implement at least the following functions:

- Read and write external data
- Convert formats between I/O data and stream data
- Transmit tuples to the input stream
- Receive tuples from the output stream

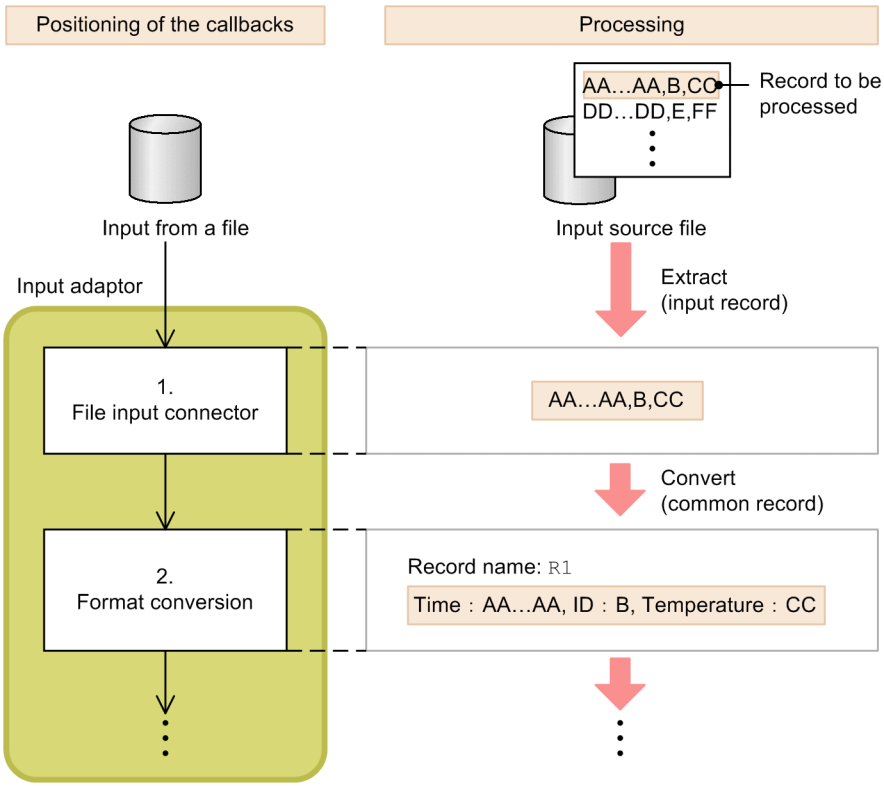
For details about custom adaptors, see the *uCosminexus Stream Data Platform - Application Framework Application Development Guide*.

3.2 Inputting files

To perform stream data processing on data files, such as log files, you use the *file input connector* as the input callback.

The file input connector extracts records to be processed from an input source file. Because these records are retrieved as input records, the format conversion callback must be used to convert them to common records so that the stream data processing engine can process them. The following figure shows the positioning and processing of the callbacks involved in file input.

Figure 3-5: Positioning and processing of the callbacks involved in file input



1. The file input connector extracts the first line (record) from the input file. The record that it extracts is called an input record.
2. The format conversion callback converts the input record to a common record.

Reference note:

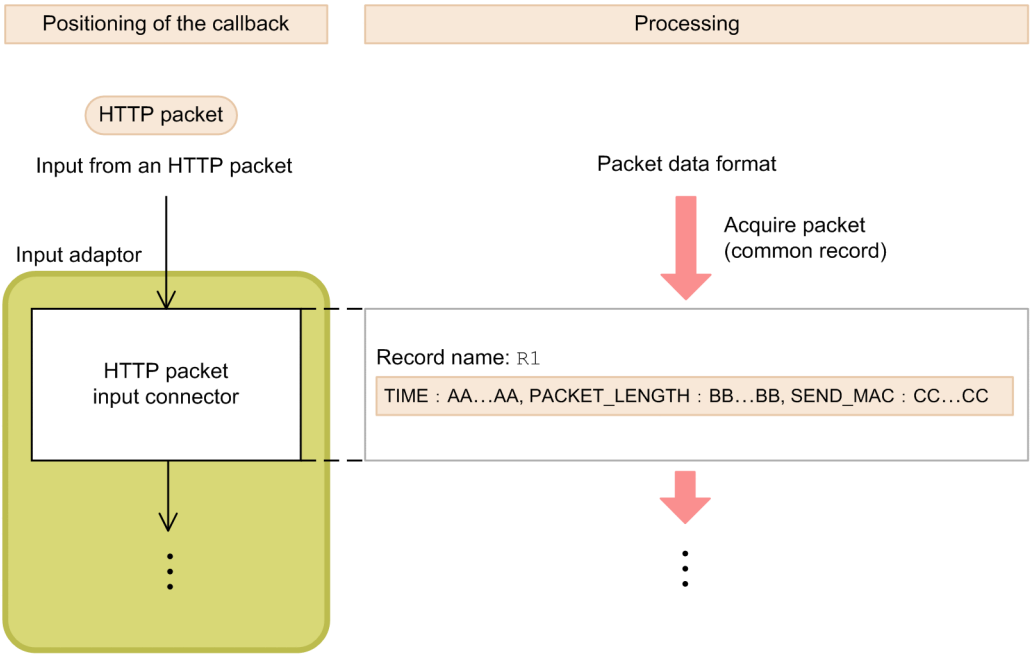
You can also extract and process multiple records at a time from the input source file.

3.3 Inputting HTTP packets

To perform stream data processing on HTTP packets carried over a network, you use the *HTTP packet input connector* as the input callback.

This packet input connector extracts HTTP packets from the output of a packet analyzer. The following figure shows the positioning and processing of the callback involved in HTTP packet input.

Figure 3-6: Positioning and processing of the callback involved in HTTP packet input



As shown in the figure, the packet input connector extracts the HTTP packet, and then converts it to a common record data format that the stream data processing engine can handle.

3.4 Filtering records

To perform stream data processing only on specific records, you use a *filter* as the data editing callback.

Reference note:

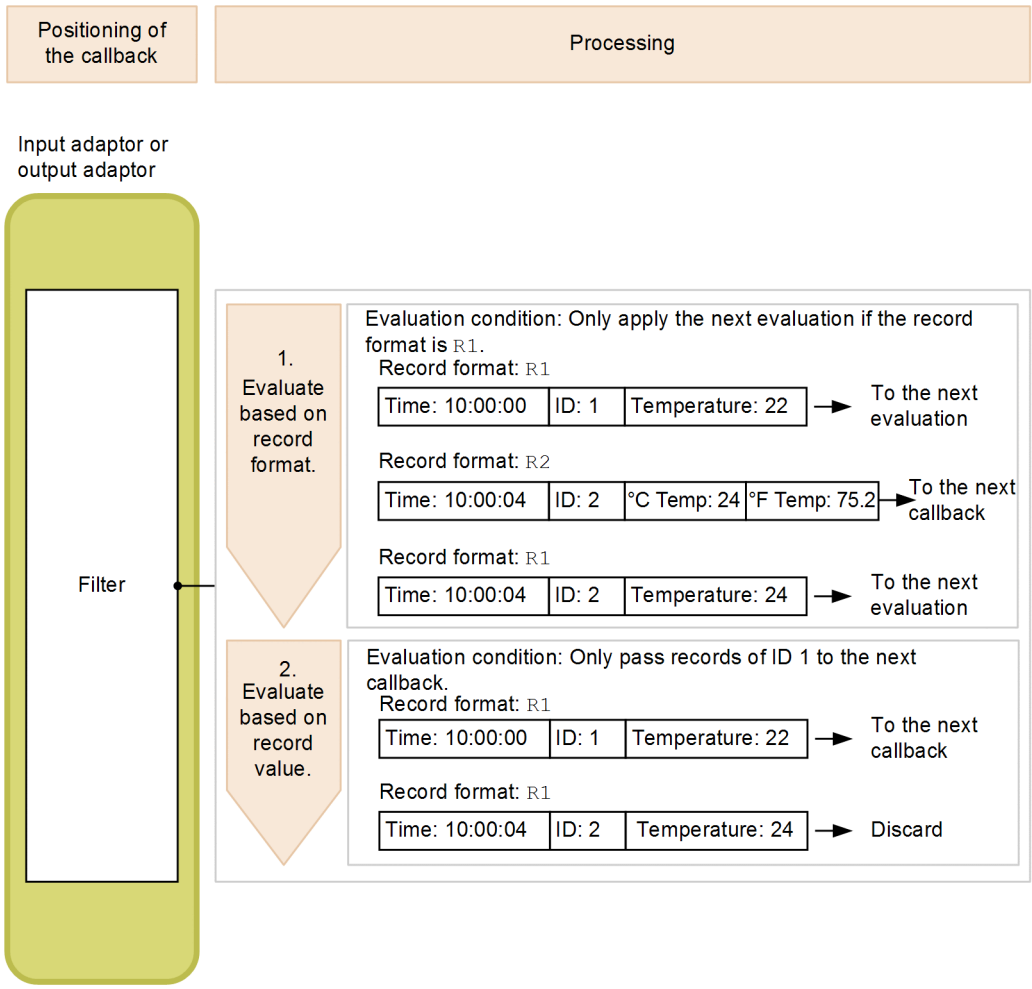
> As explained in *2.3.2 Retrieving data that satisfies a condition*, you can perform filter processing using CQL, but we recommend that you use the adaptor to improve performance.

For example, if you are monitoring temperatures from a number of observation sites and you want to summarize and analyze temperatures from only one particular observation site, you can filter on that observation site's ID.

Only common records can be filtered. If the input source is a file, after an input record is extracted by the file input connector, you must use the format conversion callback to convert it to a common record before filtering it.

When specifying the evaluation conditions you want to filter on, you can use any of the record formats and values that are defined in the records. The following figure shows the positioning and processing of the callback involved in record filtering.

Figure 3-7: Positioning and processing of the callback involved in record filtering



1. The records passed to the filter are first filtered by record format. Only records of record format `R1` meet the first condition, so only these records are selected for processing by the next condition. Records that do not satisfy this condition are passed to the next callback.
2. After the records are filtered by record format, they are then filtered by record value. This condition specifies that only those records whose ID has a value of `1` are to be passed to the next callback. In this way, only those records that satisfy both conditions will be processed by the next callback. Records that do not satisfy these conditions are discarded.

3.5 Extracting records

After you have filtered for the desired records, you use a *record extraction* callback to collect all of the necessary information from the filtered records into a single record.

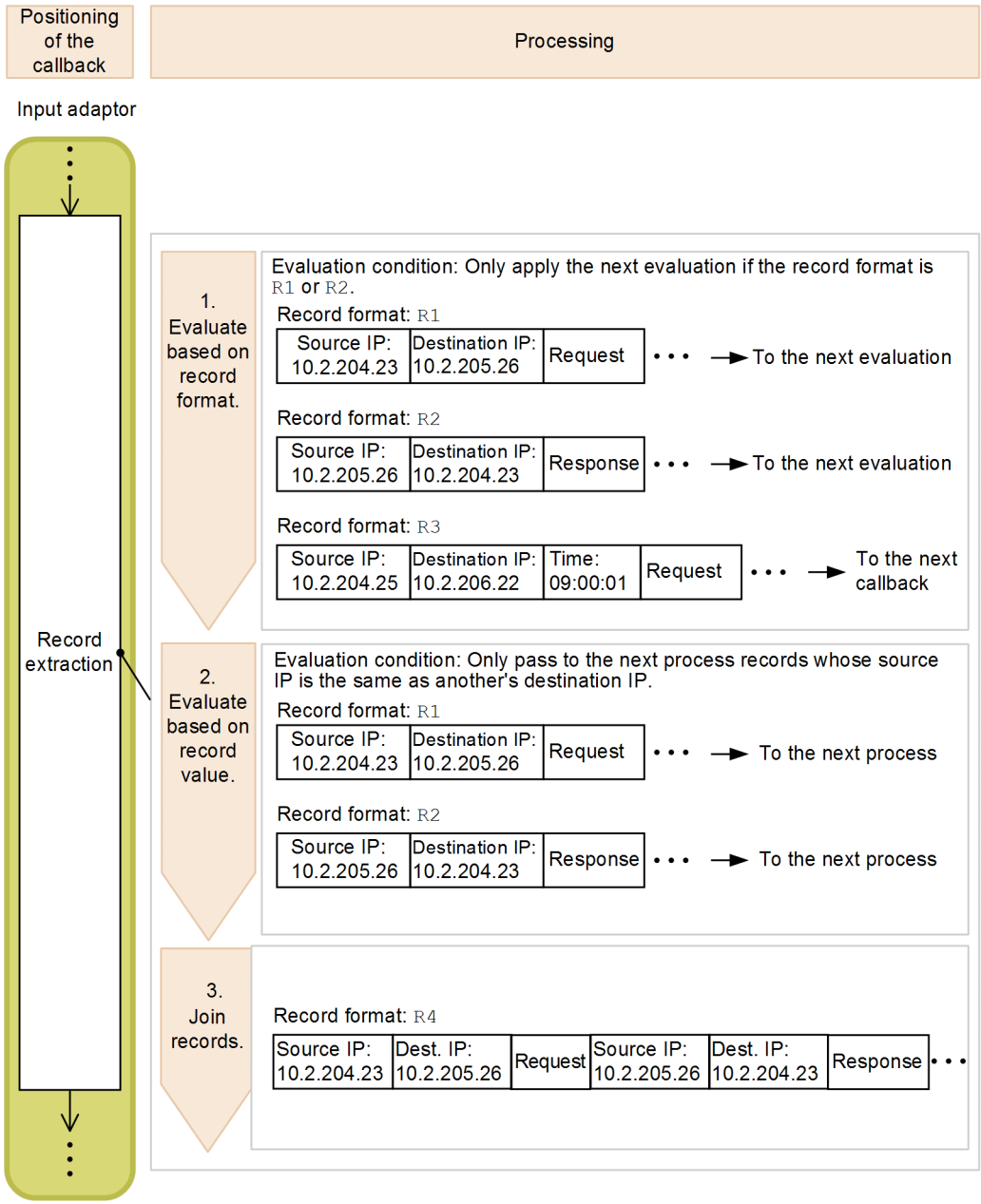
Reference note:

> As explained in *2.3.6 Joining data streams*, you can perform record extraction processing using CQL, but we recommend that you use the adaptor to improve performance.

For example, to summarize and analyze the responsiveness between a client and a server, after the HTTP packet input connector is used as the input callback, you could use a record extraction callback as the data editing callback. You could then use the record extraction callback to join an HTTP request and response packet pair into one record, based on the transmission source IP addresses and the transmission destination IP addresses. This would allow you to gain a clear understanding of response times, and to easily summarize and analyze the resulting data.

In the following figure, after records are filtered by record format and record value so that only the desired records are selected, the record extraction callback joins the resulting records, and generates a new record. The following figure shows the positioning and processing of the callback involved in record extraction.

Figure 3-8: Positioning and processing of the callback involved in record extraction



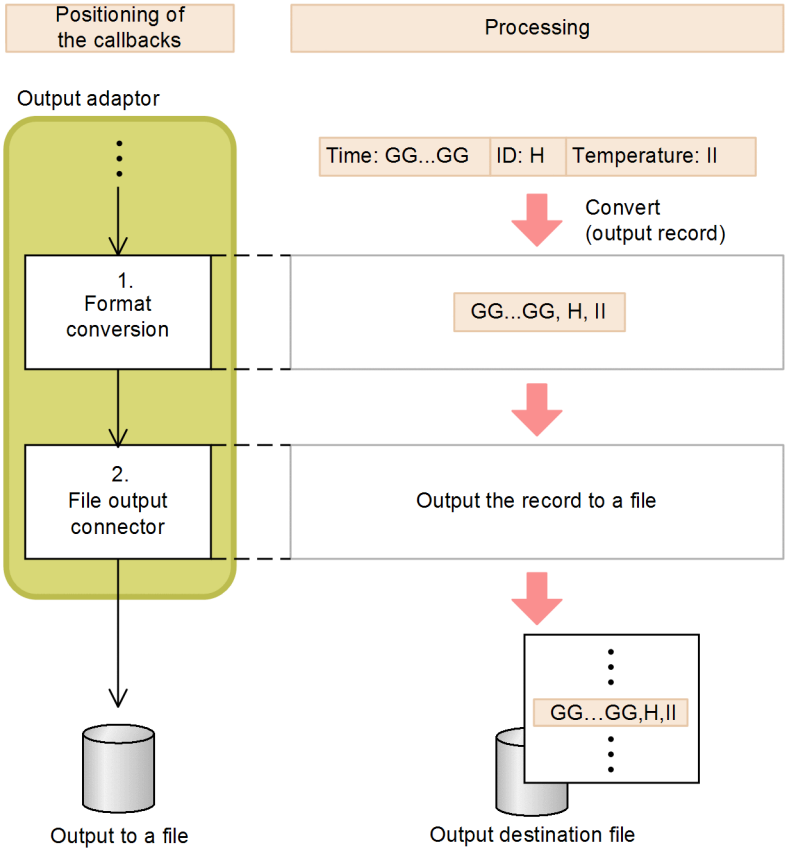
1. Records passed to the record extraction callback are first filtered by record format. Only records whose record format is `R1` or `R2` meet the first condition, so only these records are selected for processing by the next condition. Records that do not satisfy this condition are passed to the next callback.
2. After the records are filtered by record format, they are then filtered by record value. This condition specifies that records are to be passed to the next process only if the source IP of the request matches the destination IP of the response, and the destination IP of the request matches the source IP of the response. This means that only those records that match this condition are passed to the next process.
3. Records filtered by record format and record value are joined to produce a single record. Records joined in this step are selected for processing by the next callback.

3.6 Outputting to files

To output the results of stream data processing to a file, you use the *file output connector* as the output callback.

The file output connector sends output records to a file. This means that you must first use a format conversion callback to convert the common records to valid output records before running the file output connector. The following figure shows the positioning and processing of the callbacks that are involved in file output.

Figure 3-9: Positioning and processing of the callbacks involved in file output



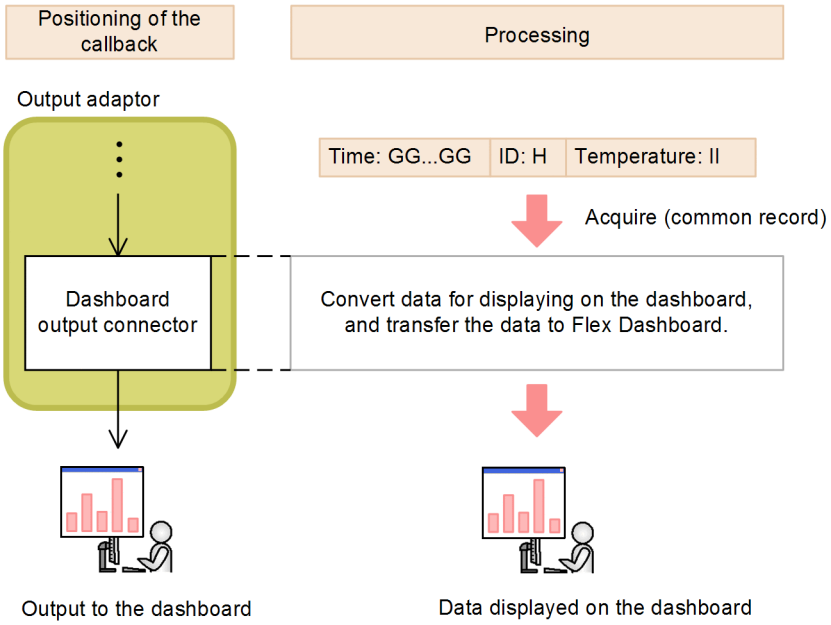
1. To output the results of stream data processing to a file, the format conversion callback first converts the record format to a valid output record format.
2. The file output connector then sends the output records to a file.

3.7 Outputting to the dashboard

To display the results of stream data processing to the dashboard, you use the *dashboard output connector* as the output callback. Data output to the dashboard can be displayed as a line chart, a bar chart, or in other chart formats.

The dashboard output connector gets common records from the previous callback. The dashboard output connector then converts these records to data that can be displayed on the dashboard, and transfers the data to Flex Dashboard. The following figure shows the positioning and processing of the callback involved in dashboard output.

Figure 3-10: Positioning and processing of the callback involved in dashboard output



3.7.1 Flex Dashboard

As its dashboard, Stream Data Platform - AF uses Flex Dashboard, which consists of the following two applications.

- Dashboard Server

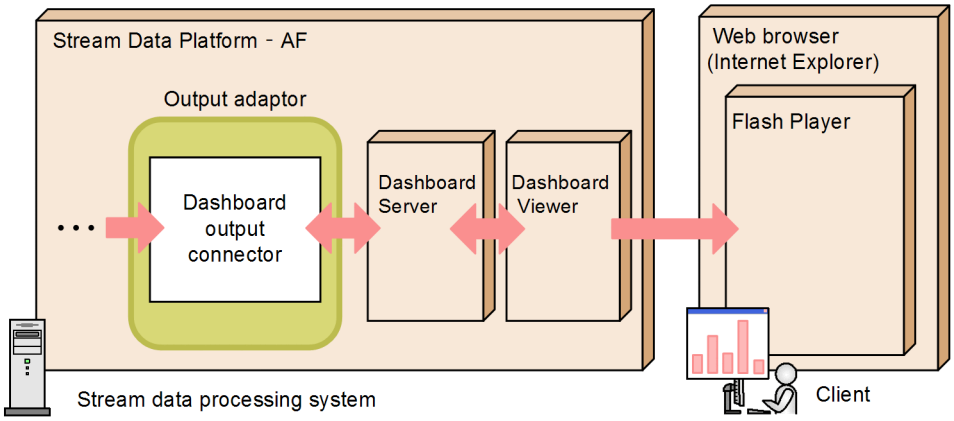
  Dashboard Server is an application that is executed by the dashboard output connector, and receives the results of stream data processing. You establish communications between the two products by entering a command on the server on which Stream Data Platform - AF runs.

- Dashboard Viewer

  Dashboard Viewer is an application that provides a GUI interface for displaying stream data processing results on a Web page. To use it, you download it to a client Web browser. The content and layout displayed by Dashboard Viewer are controlled by the definitions in the Dashboard Viewer window layout file.

The following figure shows the configuration of Flex Dashboard

Figure 3-11: Configuration of Flex Dashboard



The dashboard output connector formats the data that it receives and then passes the formatted data to Dashboard Server. Dashboard Server passes the data to Dashboard Viewer so that the browser can display the results output by the stream data processing system on the dashboard.

3.7.2 Display examples

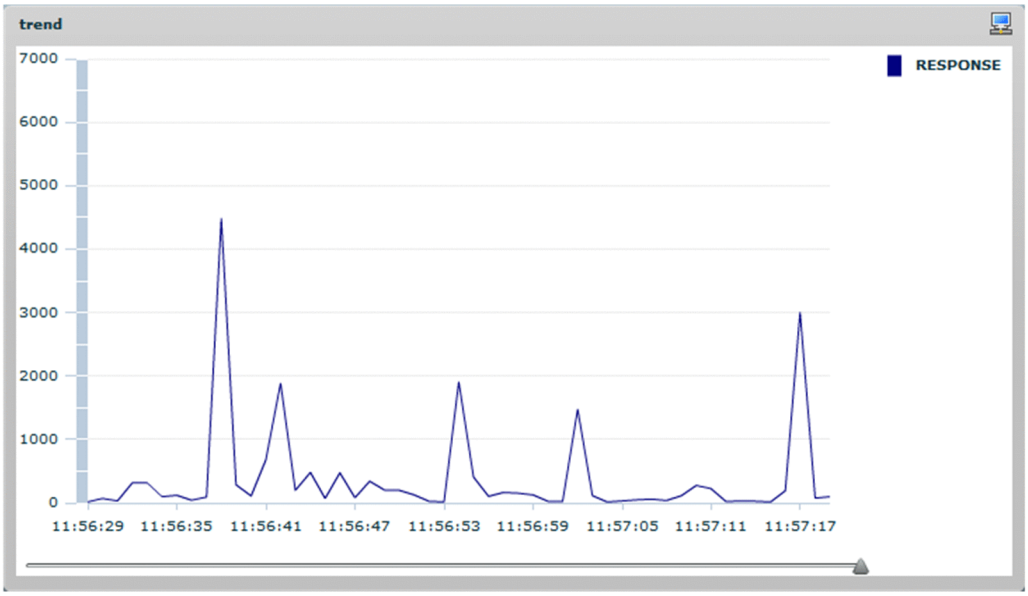
This subsection provides an overview of displaying data on the dashboard, assuming that you want to display the following summary analysis results obtained from information in HTTP packets:

- Average response time from request to response (in milliseconds)
- A breakdown of response status codes

(1) Line chart display

Using a summary analysis of HTTP packets, the following figure shows an example in which the average response time (in milliseconds) from request to response is displayed as a line chart.

Figure 3-12: Example of a display in which the average response time from request to response (in milliseconds) is presented as a line chart



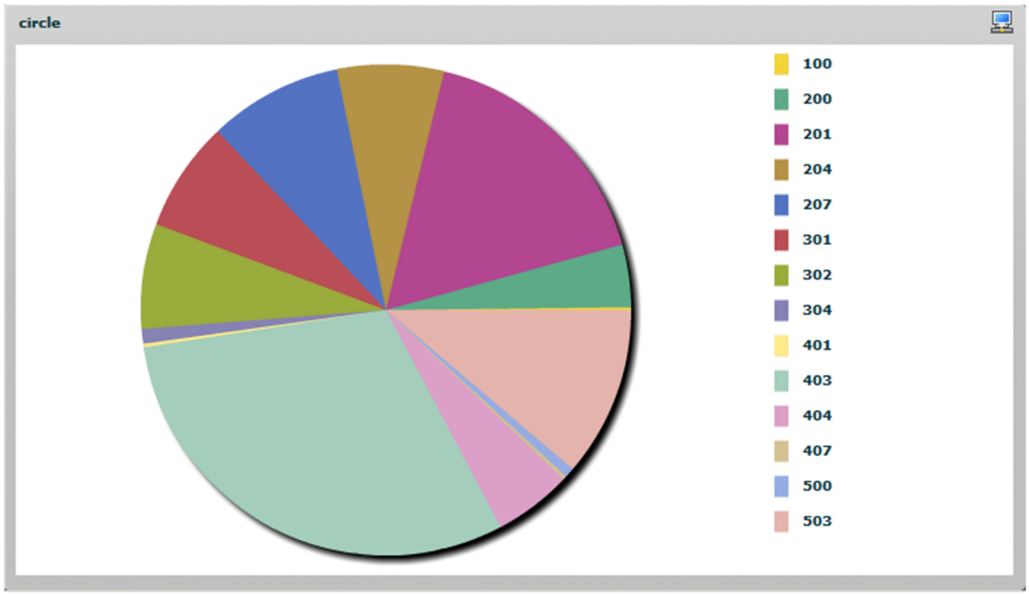
The vertical axis indicates the average response time (in milliseconds), and the horizontal axis indicates the clock time.

With the response times displayed like this, you can quickly see that, for example, the average response time from request to response at 11:57:17 was 3000 milliseconds (3 seconds).

(2) Pie chart display

Using a summary analysis of HTTP packets, the following figure shows an example in which a breakdown of response status codes is displayed as a pie chart.

Figure 3-13: Example of a display in which a breakdown of response status codes is presented as a pie chart



In the pie chart, different status codes are displayed as different colors.

With the status code breakdown displayed like this, you can quickly see that, for example, status code 403 accounts for roughly one-third of all responses.

Chapter

4. Introducing the Manuals in This Series

This chapter explains which manual in this series covers each task in the four Stream Data Platform - AF workflow phases: introduction, design, setup, and operation. This chapter also provides an overview of the manuals in the series.

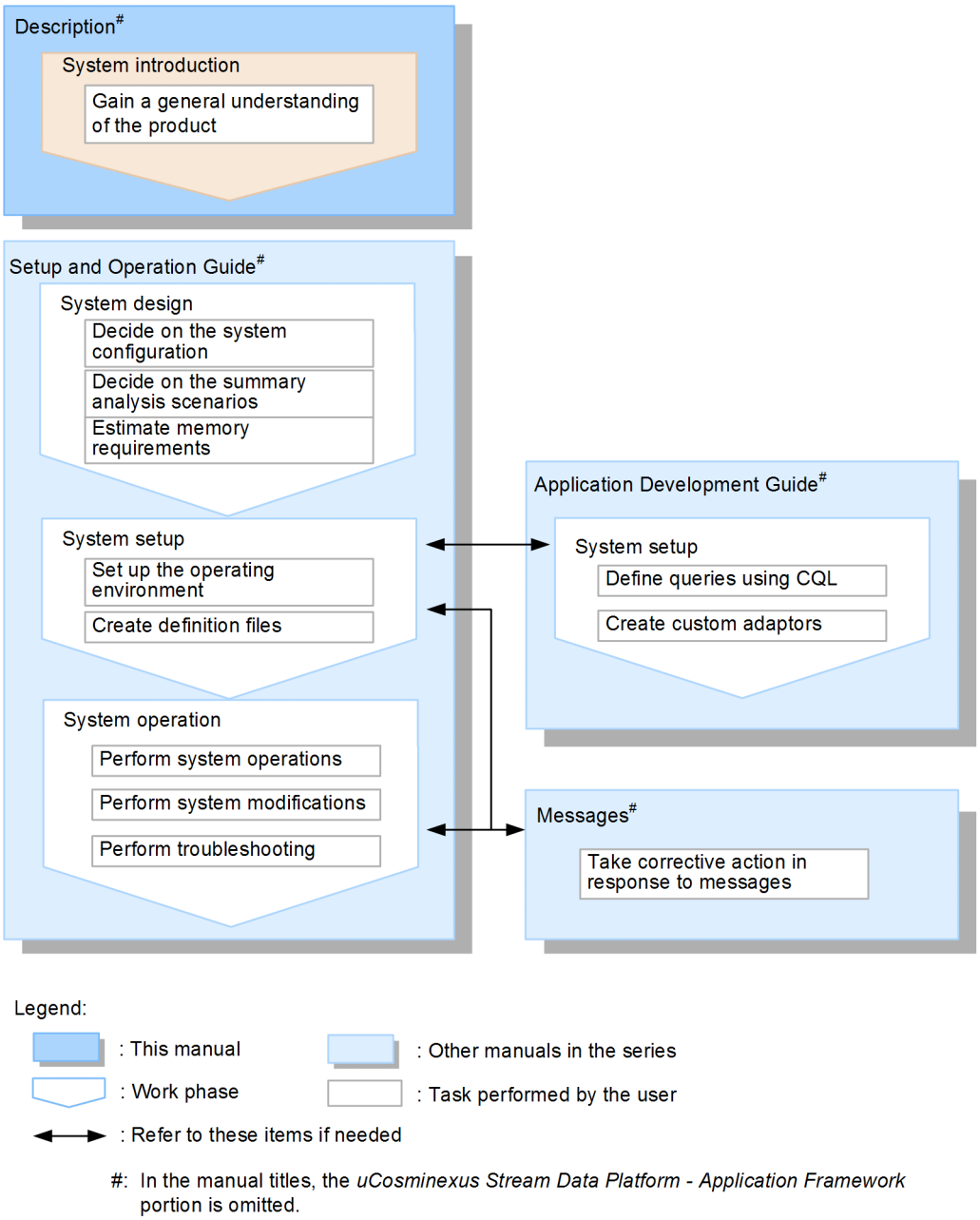
4.1 Correspondence between user tasks and the manuals in this series

The following four manuals are included in the Stream Data Platform - AF series of manuals:

- *uCosminexus Stream Data Platform - Application Framework Description*
- *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*
- *uCosminexus Stream Data Platform - Application Framework Application Development Guide*
- *uCosminexus Stream Data Platform - Application Framework Messages*

The following figure shows which manual to reference to for each task in the four Stream Data Platform - AF workflow phases: introduction, design, setup, and operation.

Figure 4-1: Correspondence between the manuals in this series and each task in the four workflow phases: introduction, design, setup, and operation



For an overview of each of these manuals, see *4.2 Overview of manuals in this series*.

4.2 Overview of manuals in this series

The following paragraphs provide an overview of each manual in the Stream Data Platform - AF series.

- *uCosminexus Stream Data Platform - Application Framework Description*

  This manual.

  We recommend that you read this manual before you implement Stream Data Platform - AF. This manual provides an overview of the functions in Stream Data Platform - AF, and a basic understanding of how to use the system.

- *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide*

  After you have read *uCosminexus Stream Data Platform - Application Framework Description*, we recommend that you read this manual next. It explains how to design, set up, and operate Stream Data Platform - AF, and it provides details about the functions that can be specified when you set up a system.

  We also recommend that you use this as a reference manual for Stream Data Platform - AF, and as an aid to troubleshooting.

- *uCosminexus Stream Data Platform - Application Framework Application Development Guide*

  After you have read *uCosminexus Stream Data Platform - Application Framework Description*, we recommend that you read this manual in the following cases:

  - If you decide during the system design phase to use custom adaptors
  - If you want to define queries using CQL

- *uCosminexus Stream Data Platform - Application Framework Messages*

  We recommend that you refer to this manual whenever Stream Data Platform - AF outputs a message, either during setup or use.

Appendixes

A. Reference Material for This Manual
B. Glossary

A. Reference Material for This Manual

This appendix provides reference information, including various conventions, for this manual.

A.1 Related publications

This manual is part of a related set of manuals. The manuals in the set are listed below (with the manual numbers):

- *uCosminexus Stream Data Platform - Application Framework Setup and Operation Guide* (3020-3-V02(E))

  This manual explains how to design, set up, and operate Stream Data Platform - AF systems, and it provides details about the functions that can be specified when you set up a system.

  We recommend that you read this manual to learn about how to analyze stream data through the design, setup, and operation of Stream Data Platform - AF systems.

- *uCosminexus Stream Data Platform - Application Framework Application Development Guide* (3020-3-V03(E))

  This manual explains how to code CQL for use in analyzing data with Stream Data Platform - AF, and how to create custom adaptors.

  We recommend that you read this manual to learn about writing CQL code for achieving analysis objectives, and to learn about using the provided APIs to create custom adaptors.

- *uCosminexus Stream Data Platform - Application Framework Messages* (3020-3-V04(E))

  This manual explains the messages output by Stream Data Platform - AF.

  We recommend that you refer to this manual if necessary when a message is output.

A.2 Conventions: Abbreviations for product names

This manual uses the following abbreviations for product names and Java-related terms:

Abbreviation	Full name or meaning
Java	Java™
JavaVM	Java Virtual Machine

Abbreviation	Full name or meaning
Stream Data Platform - AF	uCosminexus Stream Data Platform - Application Framework

A.3 Conventions: Acronyms

This manual also uses the following acronyms:

Acronym	Full name or meaning
AP	application program
API	application programming interface
CQL	Continuous Query Language
GUI	graphical user interface
HTTP	Hyper Text Transfer Protocol
IC	integrated circuit
IP	Internet Protocol
RMI	Remote Method Invocation

A.4 Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is $1{,}024^2$ bytes.
- 1 GB (gigabyte) is $1{,}024^3$ bytes.
- 1 TB (terabyte) is $1{,}024^4$ bytes.

B. Glossary

adaptor

A program required to exchange data between input sources, output destinations and the stream data processing engine.

Adaptor types include the standard adaptors provided with the product, and custom adaptors that you can program in Java.

Each of these adaptor types are further classified into input adaptors, which are used between input data and the stream data processing engine; and output adaptors, which are used between the stream data processing engine and output data.

adaptor definition file

A file used to configure the operation of standard adaptors. It specifies details about the port number used by RMI connection adaptors, the organization of the adaptor groups, and the I/O connectors used by the adaptors.

adaptor group

A group of I/O adaptors. Standard adaptors operate in adaptor groups.

Adaptor groups that implement in-process connections are called *in-process adaptor groups*. Adaptor groups that implement RMI connections are called *RMI groups*.

adaptor trace

A listing of troubleshooting-related information for tracing the process states of an adaptor.

callback

A processing unit that controls the functionality provided in the standard adaptors.

common record

An internal record format that enables records to be processed by a stream data processing system.

connector

An interface defined in the standard adaptors for connecting Stream Data Platform - AF to the outside world.

For input to Stream Data Platform - AF, the file input connector and the HTTP packet input connector are provided. For output from Stream Data Platform - AF, the file output connector and the dashboard output connector are provided.

CQL (Continuous Query Language)

A query language designed for writing continuous queries.

custom adaptor

An adaptor created by the user with the Java APIs provided by Stream Data Platform - AF.

data reception application

A client application that performs event processing on stream data output by an SDP server.

data source mode

A mode for assigning timestamps to tuples. In this mode, when the log file or other data source being input contains time information, that time information is assigned to the tuple.

data transmission application

A client application that sends stream data to an SDP server.

field

The basic unit of value in a record.

in-process connection

An architecture for connecting adaptors and SDP servers. Adaptors and SDP servers that run in the same process use an in-process connection to exchange data.

input record

A record that is read when the input source is a file.

input relation

A tuple group retrieved by means of a window operation. A relation operation is then performed on the tuple group.

intermediate relation

A tuple group retrieved by the `WHERE` clause during relation operation processing.

operator

The smallest unit of stream data processing. A query consists of one or more operators.

output record

A record format for outputting stream data processing results to a file.

output relation

A tuple group output from a relation operation. A stream operation is then performed

on the tuple group.

query

Code that defines the processing to perform on stream data. Queries are written using CQL.

query group

A stream data summary analysis scenario created in advance by the user. A query group consists of an input stream queue (input stream), an output stream queue (output stream), and relational queries.

record

A single row of data handled by stream data processing.

record organization

An organization expressed as a particular combination of two or more fields (field names and their associated values).

relation

A set of records with a given life span. Using a CQL window specification, records are converted from stream data to a relation that will persist for the amount of time specified in the window operation.

relation operation

An operation that specifies what processing is to be performed on the data retrieved by a window operation. Available actions include calculation, summarization, joining, and others.

RMI connection (Remote Method Invocation)

An architecture for connecting adaptors and SDP servers. Adaptors and SDP servers that run in different processes use Java RMI to exchange data.

SDP server

A server process running a stream data processing engine to process stream data.

SDP server definition file

A file used to configure SDP server operations. It specifies settings such as the JavaVM startup options for running an SDP server and adaptors, SDP server port numbers, and details about the API trace logs and tuple logs to acquire.

server mode

A mode for assigning timestamps to tuples. In this mode, when a tuple arrives at the stream data processing engine, the system time of the server on which Stream Data Platform - AF is running is assigned to the tuple.

standard adaptor

An adaptor provided by Stream Data Platform - AF. Standard adaptors can handle files or HTTP packets as input data, and they can output the processing results to a file, or display them on the dashboard.

stream

Data that is in a streaming (time sequence) format. Stream data that passes through an input stream queue is called an *input stream*, and stream data that passes through an output stream queue is called an *output stream*.

stream data

Large quantities of time-sequenced data that is continuously generated.

stream data processing engine

The part of a stream data processing system that actually processes stream data, as instructed by queries.

stream operation

An operation that specifies how to output data in an output relation.

stream queue

A path used for input and output of stream data. A stream queue that is used as input to the stream data processing engine is called an *input stream queue*, and a stream queue that is used as output from the stream data processing engine is called an *output stream queue*.

time division function

A function by which a `RANGE` window is partitioned into desired units of time (meshing), and the data in each of these partitioned time units is processed separately.

timestamp

The data time in a tuple.

tuple

A stream data element that consists of a combination of values and time (timestamp).

tuple log

A log file containing information on the tuples that are input to the stream data processing engine, and from tuples that are output from the stream data processing engine.

window

A range that specifies the extent of stream data that is to be summarized and analyzed. Windows are defined in queries.

window operation

An operation used to specify a window. Window operations are coded in CQL queries.

Index

U

V

W