PUBLIC

SAP HANA Platform SPS 12
Document Version: 1.0 – 2016-05-11

SAP HANA Smart Data Streaming: Developer Guide

Content

1 Introduction

SAP HANA smart data streaming makes use of Continuous Computation Language (CCL) and its components to develop streaming projects. You can use CCL through a command line interface, in SAP Web IDE for SAP HANA, or in conjunction with the smart data streaming plugin for SAP HANA studio.

If you are looking for information on building CCL projects or building applications that will create CCL projects, begin with this guide. Once you get started with development, you may also want to refer to the *SAP HANA Smart Data Streaming: CCL Reference*.

In this section:

Related Information

SAP HANA Smart Data Streaming: CCL Reference

1.1 What is Smart Data Streaming?

SAP HANA smart data streaming is a specialized option that processes streams of incoming event data in real time, and collects and acts on this information.

Smart data streaming is ideally suited for situations where data arrives as events happen, and where there is value in collecting, understanding, and acting on this data right away. Some examples of data sources that produce streams of events in real time include:

- Sensors
- Smart devices
- Web sites (click streams)
- IT systems (logs)
- Financial markets (prices)
- Social media

Data flows into streaming projects from various sources, typically through adapters, which connect the sources to the smart data streaming server. The streaming projects contain business logic, which they apply to the incoming data, typically in the form of continuous queries and rules. These streaming projects are entirely event-driven, turning the raw input streams into one or more derived streams that can be captured in the SAP HANA database, sent as alerts, posted to downstream applications, or streamed to live dashboards.

Streaming Cluster Nodes

An SAP HANA smart data streaming cluster consists of one or more nodes, with one node per host. Streaming cluster nodes run and manage the streaming projects you deploy. Streaming nodes run on dedicated hosts, one node per host. You can start with one or more streaming nodes, and add additional nodes as needed for capacity.

Streaming Workspaces

Projects are deployed in at least one workspace on a streaming cluster. A workspace provides a namespace for the project and allows you to control permissions at the workspace level.

Streaming Projects

A project defines one or more event streams and the business logic applied to incoming event data to produce results. It may also include adapters to establish connections to event sources as well as destinations (including SAP HANA tables). At its most basic level, a project consists of streams and/or windows, and adapters.

- Adapters connect a stream or window to a data source or destination.
- A stream processes incoming events without retaining and storing data, and produces output events according to an applied continuous query.
- A window receives data, but can also retain and store data. Incoming events can add, update, or delete rows in the window's table.

1.2 How SAP HANA Smart Data Streaming Works

SAP HANA smart data streaming enables you to create and run complex event processing applications to derive continuous intelligence from streaming event data in real time.

Smart data streaming is a technique for analyzing information about events, in real time, for situational awareness. When vast numbers of event messages are flooding in, it is difficult to see the big picture. With smart data streaming, you can analyze events as they stream in, and identify emerging threats and opportunities as they happen. Smart data streaming filters, aggregates, and summarizes data to enable better decision-making based on more complete and timely information.

Unlike traditional databases, which are designed for on-demand queries and transaction processing, smart data streaming is optimized for continuous queries. Thus, instead of replacing databases, it complements them to help solve new classes of problems where continuous, event-driven data analysis is required.

SAP HANA Smart Data Streaming Deployments

Data flows into the SAP HANA smart data streaming server from external sources through built-in or custom adapters, which translate incoming messages into a format that is accepted by the SAP HANA smart data streaming server.

This figure shows a typical smart data streaming deployment. Continuous queries, developed and tested as projects using the SAP HANA smart data streaming plugin for SAP HANA studio, are deployed to an SAP HANA smart data streaming server. Output adapters translate rows processed by the server into message formats that are compatible with external destinations, such as SAP HANA, and send those messages downstream. SAP HANA cockpit provides an operations console for configuring smart data streaming.

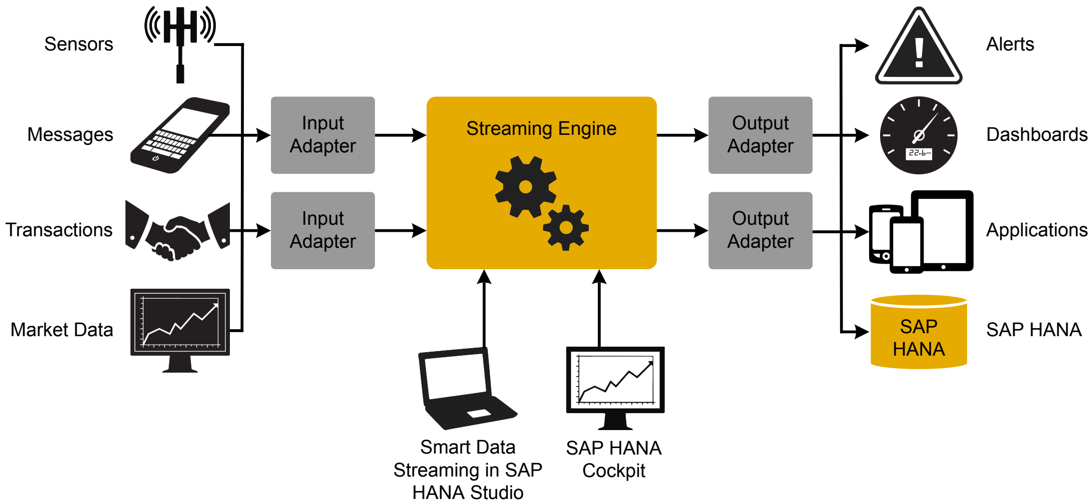


Figure 1: Smart Data Streaming Deployment

Getting Results from a Streaming Project

The output from smart data streaming can be used in several ways. These options are not mutually exclusive, and can be used in any combination.

You can:

- Push output into SAP HANA database tables, either by logging events in the tables, or by updating database tables based on the results of continuous queries.
- Send output to downstream applications to act on the information. The output can be pushed directly to an application (via an adapter), or published onto a message bus. For example, you can open an urgent work order in response to new information, or change a price based on market conditions.
- Stream output to a live operational dashboard, or other custom UI, typically via WebSockets.
- Start a new streaming project that binds (connects) to an output stream in a running project.
- Query event windows maintained in streaming projects using SAP HANA Smart Data Access.

1.3 SAP HANA Smart Data Streaming within the SAP HANA Platform

Smart data streaming absorbs incoming events delivered via streams, in real time, and at high rates. It filters and transforms data, and efficiently captures desired portions of the data in the SAP HANA database.

Smart data streaming provides active monitoring of event streams, with the ability to generate immediate alerts, notifications or responses when something happens.

Smart data streaming extends the capabilities of the SAP HANA platform by:

- Capturing data that is arriving at high speeds as individual events.
- Micro-batching and parallel processing to optimize load speeds.
- Capturing events that are published from such streaming sources as a message bus.
- Filtering, transforming or enriching incoming data so that it is in the form you require.
- Prioritizing and directing data to SAP HANA, or in other directions such as into Hadoop.
- Monitoring incoming event streams, watching for trends and patterns, and detecting missing events.
- Continuously updating and monitoring aggregate statistics.
- Generating alerts and notifications and initiating immediate responses.

Smart data streaming is not a replacement for databases. While databases excel at storing and querying static data, and reliably processing transactions, smart data streaming excels at continuously analyzing fast moving streams of data.

- Traditional databases must store all data on disk before beginning to process it.
- Smart data streaming can evaluate queries incrementally as data arrives.

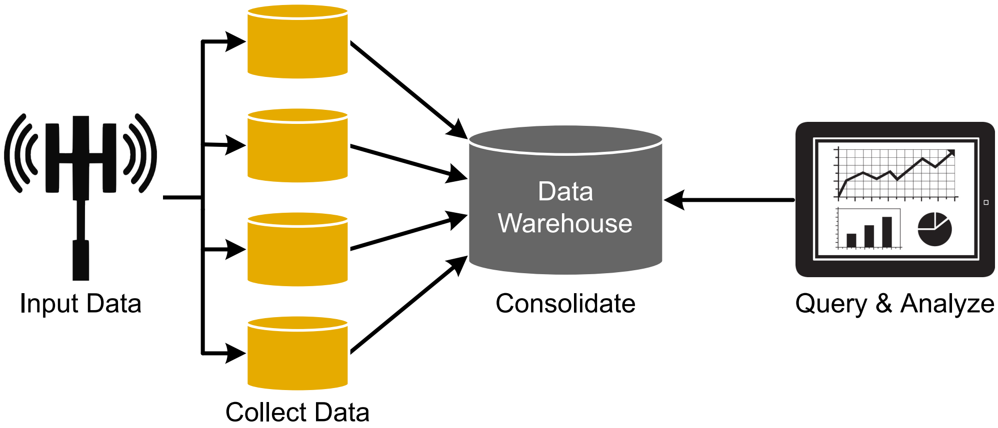


Figure 2: Traditional Business Intelligence: On-Demand Queries

Smart data streaming is not an in-memory database, although it stores all data in memory. Smart data streaming is optimized for continuous queries, rather than on-demand queries and transaction processing.

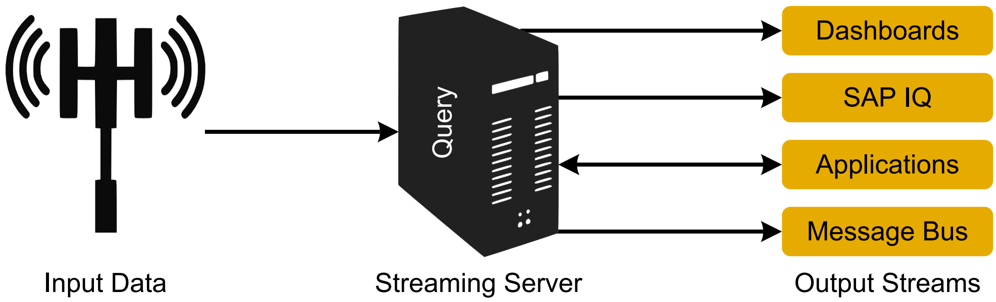


Figure 3: Query Runs Continuously on Changing Data Minimal Latency – Fast Response

1.4 Continuous Computation Language

Continuous Computation Language (CCL) is the primary event processing language of SAP HANA smart data streaming. You define projects using CCL.

CCL is based on Structured Query Language (SQL), and adapted for stream processing.

CCL supports sophisticated data selection and calculation capabilities, including features such as data grouping, aggregations, and joins. However, CCL also includes features that are required to manipulate data during real-time continuous processing, such as windows on data streams, and pattern and event-matching.

A SQL query typically executes only once each time it is submitted to a database server, and is resubmitted every time a user or an application needs to re-execute the query. By contrast, a CCL query can continuously process dynamic data, making it a key distinguishing feature. Once you define a CCL query in the project, it is registered for continuous execution and stays active indefinitely. When the project is running in the smart data streaming server, a registered query executes each time an event arrives from one of its datasources.

Although CCL borrows SQL syntax to define continuous queries, the smart data streaming server does not use a SQL query engine. Instead, it compiles CCL into a highly efficient byte code that is used by the smart data streaming server to construct continuous queries within the data-flow architecture.

CCL queries are converted to an executable form by the CCL compiler. Smart data streaming servers are optimized for incremental processing, so the query optimization is different than for databases. Compilation is typically performed within the SAP HANA studio, but you can also compile projects by invoking the CCL compiler from the command line.

CCLScript

CCLScript is a scripting language that brings extensibility to CCL, allowing you to create custom operators and functions that go beyond standard SQL. CCLScript consists of expressions used to compute values from other values, as well as variables and looping constructs, with the ability to organize instructions in functions. CCLScript syntax is similar to C and Java, though it also has similarities to languages that solve relatively small programming problems, such as AWK and Perl. CCLScript also allows you to define any complex computations using procedural logic rather than a relational paradigm. You can learn more about CCL Script in the *SAP HANA Smart Data Streaming: CCL Reference*.

Related Information

SAP HANA Smart Data Streaming: CCL Reference

1.5 Methods for Working with Streaming Projects

There are a number of ways to create, edit, and deploy SAP HANA smart data streaming projects. The most common approach is to start by using the design time tools that are available for SAP HANA.

SAP HANA studio with SAP HANA Smart Data Streaming plugin

SAP HANA studio is the Eclipse-based development environment for building applications that run on SAP HANA. The Streaming plugin for the SAP HANA studio provides two specific perspectives to create, edit, test, and deploy CCL projects: the SAP HANA Streaming Run-Test and SAP HANA Streaming Development perspectives. The SAP HANA Streaming Development perspective includes a choice of two different editors – a syntax-aware text editor and a drag-and-drop visual editor. You can easily switch between the visual editor and the text CCL editor at any time, and any changes you make in one editor are reflected in the other. The SAP HANA Streaming Run-Test perspective provides a variety of tools that make it easy to test streaming projects. You can deploy projects to the server, stream in sample data, and view output. Additional tools facilitate debugging and performance tuning. You can maintain streaming projects in the SAP HANA repository and include them in SAP HANA Delivery Units.

For more information, see Working with Projects in the Streaming Plugin for SAP HANA Studio [page 43]

SAP Web IDE for SAP HANA

With SP 12, SAP HANA smart data streaming introduces a CCL editing plugin to be used with SAP Web IDE for SAP HANA. SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) for building HANA applications. The SAP HANA smart data streaming plugin combined with SAP Web IDE for SAP HANA provide basic CCL editing capabilities, allowing users of the Web IDE to create and deploy simple CCL projects. The plugin is not a replacement for the SAP HANA studio, and does not provide the breadth of streaming design time tools that are available in the SAP HANA studio. You should continue to use SAP HANA studio as the primary development environment for SAP HANA Smart Data Streaming projects.

For more information, see Working with Projects in Web IDE [page 34].

Application-Generated Content

Applications can generate and deploy dynamic streaming content. Similar to the way applications can create database content by executing SQL DDL commands, applications can create and deploy CCL projects. An application can deploy predefined CCL projects as needed, or alternatively, an application can dynamically create and deploy CCL. The SAP HANA smart data streaming server provides a REST interface which applications can use to deploy CCL to the server. Alternatively, there is a Java library which you can use for applications written in Java, and have the ability to connect to the SAP HANA system via TCP sockets. For Java programmers, there is also a library to read and write CCL files.

Text Editors, Command Line Tools and Other Methods

Because streaming projects are simple text files, you can also use any editor to create, view and edit CCL projects. You can compile and deploy through the command line tools (which are documented in the *SAP HANA Smart Data Streaming: Utilities Guide*), the REST interface provided by the smart data streaming server, or the Java API.

Related Information

SAP HANA Smart Data Streaming: Utilities Guide

2 CCL Project Basics

Smart data streaming projects are written in CCL, a SQL-like language that specifies a data flow (by defining streams, windows, operations, and connections), and provides the ability to incorporate functions written in other languages, such as CCLScript, to handle more complex computational work.

Projects consist of a set of event streams, other required datasources, and the business logic applied to incoming event data to produce results. At its most basic level, a project consists of the following elements:

Element	Description
Input streams and windows	Receive input data flowing into the project. An input stream can receive: • Incoming event data on an event-driven basis. • Static or semistatic sets of data that are loaded once or periodically refreshed. Input streams that have state—that is, they can retain and store data—are called windows.
Adapters	Connect an input stream or window to a datasource, and connect an output stream or window to a destination. Smart data streaming includes: • An extensive set of input and output adapters to subscribe to and publish data. • An adapter toolkit for building custom external (Java) adapters. • A variety of SDKs to build custom external adapters in a number of programming languages. While an adapter connects the project to external inputs and outputs, technically it is not part of the project.
Derived streams and windows	Take data from one or more streams or windows and apply a continuous query to produce a new stream or window. Derived streams that have state—that is, they can retain and store data—are called windows.

All project information is stored in the project's CCL file, `<project-name>.ccl`, which is the definition of the project in Continuous Computation Language. The project also has one or more CCR files. These project configuration (.ccr) files are XML files that contain runtime and deployment configuration for the project, including adapter, parameter, and binding definitions. A project can have multiple CCR files allowing you to specify different adapters in different deployment environments, or to change runtime parameter values.

In this section:

Events [page 17]

An event is a message that contains information about an actual business event that occurred. Creating SAP HANA smart data streaming allows you to consume and process input events and generate output events.

Operation Codes [page 18]
The operation code (opcode) of an event record specifies the action to perform on the underlying store of a window for that event.

Streams and Windows [page 19]
Both streams and windows can process events. The difference is that windows have state, meaning they can retain events, and can apply operations across the set of events in the window, while streams are stateless, and only operate on the current event. Streams and windows offer different characteristics and features, but also share common designation, visibility, and column parameters.

Schemas [page 30]
A schema defines the structure of data rows in a stream or window.

Adapters [page 30]
Adapters connect SAP HANA smart data streaming to the external world.

SAP HANA Reference Tables [page 31]
Reference table queries help augment the streaming data in a SAP HANA smart data streaming project with information from a table or view in SAP HANA.

CCL Functions [page 32]
A function is a self-contained, reusable block of code that performs a specific task.

Queries in CCL [page 32]
CCL queries are attached to derived streams or windows to select data from one or more inputs and transform it into the desired output.

Bindings between CCL Projects [page 33]
Bindings enable data to flow between projects by allowing a stream, keyed stream, or window in one project to subscribe or publish to a stream, keyed stream, or window in another project.

2.1 Events

An event is a message that contains information about an actual business event that occurred. Creating SAP HANA smart data streaming allows you to consume and process input events and generate output events.

In smart data streaming, an event is represented by a set of typed fields and an operation code.

Examples of business events that are often transmitted as streams of event messages include:

- Financial market data feeds, which transmit trade and quote events. Each event could consist of ticket symbol, price, quantity,and time.
- Radio Frequency Identification System (RFID) sensors, which transmit events indicating that an RFID tag was sensed nearby.
- Electronic sensors, which transmit messages indicating the health of remote equipment and its components.
- Click streams, which transmit a message (a click event) each time a user clicks a link or button on a website.

- Database transaction events, which occur each time a record is added or updated.

In this example, a project monitors streams of financial market data. The stream `StreamTrades` accepts an `insert` event, which populates the columns `Ts`, `Symbol`, `Price`, and `Volume`:

```
StreamTrades ESP_OPS="insert" Ts="2012" Symbol="DRCT" Price="30.4400"
Volume="100"
```

Event Blocks

You can publish business events into smart data streaming projects in collections called event blocks, which improve the performance of your projects. Event blocks come in two different types: envelopes and transactions. As an event block is being processed by a window, resulting rows are stored until the last event of the block is processed, after which the resulting events are sent downstream.

Event blocks have the following properties:

- Envelopes:
  - Each row in an envelope is treated atomically; a failure in an event does not discard the envelope. This behavior is useful if a model's performance is important, but not necessarily the integrity of the data.
- Transactions:
  - A transaction is discarded if any one event in the block fails. Use this behavior to guarantee that logical blocks of events are completely error-free.
  - Before a transaction block is sent downstream, all events in the transaction are compressed as much as possible. For example, an event with an insert and then an update will compress down to a single insert with updated values.

2.2 Operation Codes

The operation code (opcode) of an event record specifies the action to perform on the underlying store of a window for that event.

In many smart data streaming use cases, events are independent of each other; each carries information about something that happened. In these cases, a stream of events is a series of independent events. If you define a window on this type of event stream, each incoming event is inserted into the window. If you think of a window as a table, the new event is added to the window as a new row.

In other use cases, events deliver new information about previous events. Smart data streaming maintains a current view of the set of information as the incoming events continuously update it. Two common examples are ordering books for securities in capital markets, and opening orders in a fulfillment system. In both applications, incoming events may indicate the need to do one of the following:

- Add an order to the set of open orders
- Update the status of an existing open order
- Remove a canceled or filled order from the set of open orders

To handle information sets that are updated by incoming events, smart data streaming recognizes the following opcodes in incoming event records:

- `insert`: inserts the event record.
- `update`: updates the event record with the specified key. If no such record exists, it prompts a runtime error.
- `delete`: deletes the record with the specified key. If no such record exists, it prompts a runtime error.
- `upsert`: updates the record with the matching key. If no such record exists, it inserts the record.
- `safedelete`: deletes the record with the matching key. If no such record exists, it does nothing.

All event records include an opcode. Each stream or window in the project accepts incoming event records and produces event records. Output events, including opcodes, are determined by their source (window, stream, or keyed stream) and the processing specified for it.

2.3 Streams and Windows

Both streams and windows can process events. The difference is that windows have state, meaning they can retain events, and can apply operations across the set of events in the window, while streams are stateless, and only operate on the current event. Streams and windows offer different characteristics and features, but also share common designation, visibility, and column parameters.

Streams process incoming events and produce output events according to the continuous query that is attached to the stream, but no data is retained.

A window consists of a table where incoming events can add, update, or delete rows. You can set the size of the window based on time or the number of events recorded. For example, a window might retain all events from the past 20 minutes, or the most recent 1,000 events. A window can also retain all events. In this case, the incoming event stream is self-managing, in that it contains events that both insert rows into the window and delete rows from the window, so that the window does not grow infinitely large. Windows are needed to perform aggregate operations and joins, since you cannot do this on streams. Although you can join a stream to a window, or join a window to another window, you can't join two streams together, because they are stateless: an event in one stream would not have anything to join with in the other stream.

Windows behave like streams in that they actively emit events (based on the input events being applied to the window). Windows can also emit events in the absence of input: a time-based retention policy on a window causes the window to emit "delete events" as rows in the window age-out.

All windows require a primary key, and every row in the window has a unique primary key value. The key value and opcode of an incoming event determine how the event is applied to the window, whether it is added as a new row, or updates or deletes an existing row in the window. Streams, on the other hand, do not usually require a primary key. If a primary key is defined on a stream, then we refer to the stream as a "keyed stream", because the presence of the primary key designation changes the behavior of the stream with respect to opcodes. The following table compares feature capabilities between streams and windows:

Feature Capability	Streams	Windows
Data retention	None	Yes, rows (based on retention policy).
Available store types	Not applicable	Memory store or log store. Windows are assigned to a memory store by default. The state of windows assigned to log stores is restored after a project restart.

Feature Capability	Streams	Windows
Primary key required	No. However, a primary key can be designated (optional).	Yes, explicit or deduced.
Support for aggregation operations	No	Yes
Behavior on receiving insert, update, or delete	• Unkeyed stream: Produces insert. Converts update to insert. Ignores delete. • Keyed stream: Opcodes are passed unchanged. Does not detect duplicate inserts, or any updates/deletes against non-existent key values.	Produces insert, update, or delete according to the OpCode Rules for Window listed below.
Behavior on receiving upsert or safedelete	• Unkeyed stream: Converts upsert to insert. Ignores safedelete. • Keyed stream: Reject upserts and safedeletes as bad events.	Produces insert, update, or delete according to the exceptions listed below.
Filter semantics	• Unkeyed stream: Assumes all events to be inserts and emits only inserts for all events that satisfy the filter condition and have insert, update or upsert opcodes. Ignores events with delete and safedelete opcodes. • Keyed stream: Emits the incoming event if it passes the filter condition without regard to, and without altering, the opcode.	Modfies the opcode according to whether the previous event with the same key passed or failed the filter.
Can directly feed a window	Yes, but requires aggregation or the use of `nextval()` or `uniqueval()` functions to produce a primary key for the window.	Yes
Can serve as a project input element	Yes	Yes
Can serve as a project output element	Yes	Yes

Opcode Rules for Smart Data Streaming Windows

A window can emit events with different opcodes, depending on what changes are made to the window contents by an incoming event. For example:

- At the most basic level, an incoming event is applied to the window, and the window emits an event that reflects the change to the window.
- In a window performing aggregation logic, an incoming event record with an insert opcode can update the contents of the store, and output an event record with an update opcode. This can also happen in a window implementing a left join.

- In a window with a count-based retention policy, an incoming event record with an insert opcode can cause the store to exceed this count. The window deletes the excess row, producing an event record with a delete opcode, and an insert event with a new row.
- An error is generated on duplicate inserts, bad updates, and bad deletes.
- Windows with a retention policy treat an update as an upsert, since the row being updated could have been removed from the window because of the retention policy.

Derived Streams and Windows

A derived stream or window is a the result of taking one or more streams or windows as input, and applying a continuous query. This means either a SELECT clause in a CREATE STREAM or CREATE WINDOW statement, or a Flex operator applying CCLScript to events that arrive from one or more input streams or windows. Derived streams either direct data locally within a project, or publish the data to external destinations.

In this section:

Related Information

Specifying a Retention Policy [page 87]

2.3.1 Streams

Streams subscribe to incoming events and process the event data according to the rules you specify (which you can think of as a "continuous query") to publish output events. Because they are stateless, they cannot retain data, and they use little memory because they do not store events.

Streams can be designated as input or derived. Input streams are the point at which data enters the project from external sources via adapters. A project may have any number of input streams. Input streams do not have continuous queries attached to them, although you can define filters for them. Derived streams either direct data locally within a project, or publish it to external destinations.

Because a stream does not have an underlying store, the only thing it can do with arriving input events is to insert them. Insert, update, and upsert opcodes are all treated as inserts. Delete and safedelete are ignored. The only opcode that a stream can include in output event records is insert.

If you specify a key on a stream, the stream's opcode handling semantics change, and it becomes a keyed stream. A keyed stream can pass insert, update, and delete events with the opcode unchanged, but will treat them all the same.

Local and output streams take their input from other streams or windows, rather than from adapters, and they apply a continuous query to produce their output. Local streams are identical to output streams, except that local streams are hidden from outside subscribers. Thus, a subscriber cannot subscribe to a local stream. You cannot monitor or subscribe to local streams in the smart data streaming plugin for SAP HANA studio.

Each subscribed stream has a single thread that posts rows to all clients. If one subscriber to this stream backs up and the client's queue is filled, it blocks subscription for all other clients.

2.3.2 Windows

A window is a stateful element that can be named or unnamed, and retains rows based on a defined retention policy.

Create a window if you need data to retain state. To create a window, you can:

- Open the **Streams and Windows** compartment of the visual editor in SAP HANA studio SAP HANA Streaming Development perspective, and click **Input Window**.
- Use the CREATE WINDOW statement in the text editor. When creating the window, and to retain rows, assign a primary key

Since a window is a stateful element, with an underlying store, it can perform any operation specified by the opcode of an incoming event record. Depending on what changes are made to the contents of the store by the incoming event and its opcode, a window can produce output event records with different opcodes.

For example, if the window is performing aggregation logic, an incoming event record with an insert opcode can update the contents of the store and thus produce an event record with an update opcode. The same could happen in a window implementing a left join.

A window can produce an output event record with the same opcode as the input event record. For example, if a window implements a simple copy or a filter without any additional clauses, the input and output event records have the same opcode.

An incoming event record with an insert opcode can produce an output event record with a delete opcode. For example, a window with a count-based retention policy (for example, keeping five records) will delete those records from the store when the sixth event arrives, thus producing an output event record with a delete opcode.

Each subscribed window has a single thread that posts rows to all clients. If one subscriber to this window backs up and the client's queue is filled, it blocks subscription for all other clients.

Named Windows

Named windows can be derived or input:

- Derived windows can be:
  - Output, which can send data to an adapter.
  - Local windows, which are private and invisible externally. If a window does not have a qualifier, it is considered local.
- Input, which can send and receive data through adapters.

Both input and output windows are visible externally and can be subscribed to or queried.

The following table compares input, output, and local windows:

Type	Receives Data From	Sends Data To	Visible Externally
input	Input adapter or external application that sends data into smart data streaming using the smart data streaming SDK.	One or more of the following: other windows, keyed streams, and output adapters.	Yes
output	Other windows, streams, or keyed streams.	One or more of the following: other windows, keyed streams, and output adapters.	Yes
local	Other windows, streams, or keyed streams.	Other windows or keyed streams.	No

Unnamed Windows

Unnamed windows are implicitly created:

- When using a join with a window that produces a stream.
- When the KEEP clause is used with the FROM clause of a statement.

In both situations, when an unnamed window is created, it always includes a primary key.

> **Note**
>
> Although unnamed windows use additional memory, there is no memory reporting on them.

This example creates an unnamed window when using a join with a window:

```
CREATE INPUT WINDOW Win1 SCHEMA (Key1 INTEGER, Col1 STRING, Col2 STRING) PRIMARY
KEY (Key1);
```

```
CREATE INPUT WINDOW Win2 SCHEMA (Key1 STRING, Col3 STRING) PRIMARY KEY (Key1);
CREATE OUTPUT WINDOW Out1 PRIMARY KEY DEDUCED AS SELECT Win1.Key1, Win1.Col1,
Win1.Col2, Win.2.Col3
FROM Win1 INNER JOIN Win2 ON Win1.Col1 = Win2.Key1;
```

> **Note**
>
> The unnamed window is created to ensure that a join does not see records that have not yet arrived at the join. This can happen because the source to the join and the join itself are running in separate threads.

These four examples demonstrate when an unnamed window is created using a KEEP clause.

Example 1 creates an unnamed window on the input `Trades` for the `MaxTradePrice` window to keep track of a maximum trade price for all symbols seen within the last 10,000 trades:

```
CREATE WINDOW MaxTradePrice
PRIMARY KEY DEDUCED
STORE S1
AS SELECT
        trd.Symbol, max(trd.Price) MaxPrice
FROM Trades trd KEEP 10000 ROWS
GROUP BY trd.Symbol;
```

Example 2 creates an unnamed window on `Trades`, and `MaxTradePrice` keeps track of the maximum trade price for all the symbols during the last 10 minutes of trades:

```
CREATE WINDOW MaxTradePrice
PRIMARY KEY DEDUCED
STORE S1
AS SELECT
        trd.Symbol, max(trd.Price) MaxPrice
FROM Trades trd KEEP 10 MINUTES
GROUP BY trd.Symbol;
```

Example 3 creates an unnamed window when using a window on a stream:

```
CREATE INPUT STREAM DataIn
SCHEMA (Symbol string, price money(2), size integer);
CREATE OUTPUT WINDOW MovingAvg
PRIMARY KEY DEDUCED AS SELECT DataIn.Symbol Symbol ,
        avg(DataIn.price) AvgPrice ,
        sum(DataIn.size) TotSize
FROM DataIn KEEP 5 MIN
GROUP BY DataIn.Symbol;
```

Example 4 creates a `FiveMinuteVWAP` unnamed window from the source stream `Trades`. Since the stream is an input to an aggregation, the unnamed window is created to allow the stream to have a retention policy:

```
CREATE INPUT STREAM Trades
SCHEMA (Tradeid integer, Symbol string, Price money(2), Shares integer)
CREATE WINDOW FiveMinuteVWAP
PRIMARY KEY DEDUCED AS SELECT
        trd.Symbol, trd.Price, trd.Shares,
        vwap(trd.Price, trd.Shares)
FROM Trades KEEP 5 MINUTES
GROUP BY trd.Symbol;
```

2.3.3 Retention Policies

A retention policy specifies the maximum number of rows or the maximum period of time that data is retained in a window.

In CCL, you can specify a retention policy when defining a window. You can also create an unnamed window by specifying a retention policy on a window or keyed stream when it is used as a source to another element.

Retention is specified through the KEEP clause. You can limit the number of records in a window based on either the number, or age, of records in the window. These methods are referred to as count-based retention and time-based retention, respectively. You can also use the ALL modifier to explicitly specify that the window should retain all records.

Caution

If you do not specify a retention policy, the window retains all records. This can be dangerous: the window can keep growing until all memory is used and the system shuts down. The only time you should have a window without a KEEP clause is if you know that the window size will be limited by incoming delete events.

Including the EVERY modifier in the KEEP clause produces a jumping window, which deletes all of the retained rows when the time interval expires or a row arrives that would exceed the maximum number of rows.

Specifying the KEEP clause with no modifier produces a sliding window, which deletes individual rows once a maximum age is reached or the maximum number of rows are retained.

Note

You can specify retention on input windows (or windows where data is copied directly from its source) using either log file-based stores or memory-based stores. For other windows, you can only specify retention on windows with memory-based stores.

Count-based Retention

In a count-based policy, a constant integer specifies the maximum number of rows retained in the window. You can use parameters in the count expression.

A count-based policy also defines an optional SLACK value, which can enhance performance by requiring less frequent cleaning of memory stores. A SLACK value accomplishes this by ensuring that there are no more than `<N>` + `<S>` rows in the window, where `<N>` is the retention size and `<S>` is the SLACK value. When the window reaches `<N>` + `<S>` rows, the system purges `<S>` rows. The larger the SLACK value, the better the performance, since there is less cleaning required.

Note

You cannot use the SLACK value in a jumping window's retention policy, because it cannot be used with the EVERY modifier.

The default value for SLACK, 1, causes a significant impact on performance, because every new inserted record deletes the oldest record. Larger SLACK values improve performance by reducing the need to constantly delete rows.

Count-based retention policies can also support retention based on content/column values using the PER sub-clause. A PER sub-clause can contain an individual column or a comma-delimited list of columns. A column can only be used once in a PER sub-clause. Specifying the primary key or autogenerate columns as a column in the PER sub-clause results in a compiler warning, because these are unique entities for which multiple values cannot be retained.

Example

The first example creates a sliding window that retains the 100 most recent records that match the filter condition:

```
CREATE WINDOW Last100Trades PRIMARY KEY DEDUCED
KEEP 100 ROWS
AS SELECT * FROM Trades
WHERE Trades.Volume > 1000;
```

Once there are 100 records in the window, the arrival of a new record deletes of the oldest record in the window.

Adding a SLACK value of 10 means the window may contain as many as 110 records before any records are deleted:

```
CREATE WINDOW Last100Trades PRIMARY KEY DEDUCED
KEEP 100 ROWS SLACK 10
AS SELECT * FROM Trades
WHERE Trades.Volume > 1000;
```

The next example creates a jumping window named `TotalCost` from the source stream `Trades`. This window retains a maximum of 10 rows, and deletes all 10 retained rows on the arrival of a new row:

```
CREATE WINDOW TotalCost
PRIMARY KEY DEDUCED
AS SELECT
        trd.*,
        trd.Price * trd.Size TotalCst
FROM Trades trd
KEEP EVERY 10 ROWS;
```

This example creates a sliding window that retains two rows for each unique value of `Symbol`. Once two records have been stored for any unique `Symbol` value, the arrival of a third record (with the same value) deletes the oldest stored record with the same `Symbol` value:

```
CREATE SCHEMA TradesSchema (
        Id integer,
        TradeTime seconddate,
        Venue string,
        Symbol string,
        Price float,
        Shares integer );
CREATE INPUT WINDOW TradesWin1
    SCHEMA TradesSchema
    PRIMARY KEY(Id)
    KEEP 2 ROWS PER(Symbol);
```

Time-based Retention

In a sliding window's time-based policy, a constant interval expression specifies the maximum age of the rows retained in the window. In a jumping window's time-based retention policy, all the rows produced in the specified time interval are deleted after the interval has expired.

Example

This example creates a sliding window that retains each record received for 10 minutes. Each individual row that exceeds the retention time limit of 10 minutes is deleted:

```
CREATE WINDOW RecentPositions PRIMARY KEY DEDUCED
KEEP 10 MINS
AS SELECT * FROM Positions;
```

The next example creates a jumping window named `Win1` that keeps every row that arrives within a 100-second interval. When the time interval expires, all of the rows retained are deleted:

```
CREATE WINDOW Win1
PRIMARY KEY DEDUCED
AS SELECT * FROM Source1
KEEP EVERY 100 SECONDS;
```

The PER sub-clause supports content-based data retention, where data is retained for a specific time interval for each unique column value or combination. A PER sub-clause can contain a single column or a comma-delimited list of columns, but you may only use each column once in the same PER clause.

Note

Time-based windows retain data for a specified time regardless of their grouping.

This example creates a jumping window that retains five seconds worth of data for each unique value of `Symbol`:

```
CREATE SCHEMA TradesSchema (
        Id integer,
        TradeTime seconddate,
        Venue string,
        Symbol string,
        Price float,
        Shares integer );
CREATE INPUT WINDOW TradesWin2
    SCHEMA TradesSchema
    PRIMARY KEY(Id)
    KEEP EVERY 5 SECONDS PER(Symbol);
```

Retention Semantics

When the insertion of one or more new rows into a window triggers the deletion of preexisting rows (due to retention), the window propagates the inserted and deleted rows downstream to relevant streams and subscribers. However, the inserted rows are placed before the deleted rows, since the inserts trigger the deletes.

Aging Policy

You can set an aging policy to flag records that have not been updated within a defined interval. This is useful for detecting records that may be stale. Aging policies are an advanced, optional feature for a window or other stateful element.

2.3.4 Input, Output, and Local

You can designate streams, windows, and keyed streams as input or derived. Derived streams and windows can be designated as either output or local.

Input and Output Streams and Windows

Input streams and windows can accept data from a source external to the project using an input adapter or by connecting to an external publisher. You can attach an output adapter or connect external subscribers directly to an input window or input stream. You can also use an ad-hoc query to select rows from an input window, insert rows in an input stream or insert, update, or delete rows in an input window.

Output windows, streams, and keyed streams can publish data to an output adapter or an external subscriber. You can use an ad-hoc query to select rows from an output window.

Local streams, windows, and keyed streams are invisible outside the project and cannot have input or output adapters attached to them. You cannot subscribe to or use an ad-hoc query for the contents of local streams, windows, or keyed streams.

Example

This is an input stream with a filter:

```
CREATE SCHEMA mySchema (Col1 INTEGER, Col2 STRING);
CREATE INPUT STREAM IStr2 SCHEMA mySchema
    WHERE IStr2.Col2='abcd';
```

This is an output stream:

```
CREATE OUTPUT STREAM OStr1
    AS SELECT A.Col1 col1, A.Col2 col2
    FROM IStr1 A;
```

This is an input window:

```
CREATE SCHEMA mySchema (Col1 INTEGER, Col2 STRING);
CREATE MEMORY STORE myStore;
CREATE INPUT WINDOW IWin1 SCHEMA mySchema
    PRIMARY KEY(Col1)
    STORE myStore;
```

This is an output window:

```
CREATE SCHEMA mySchema (Col1 INTEGER, Col2 STRING);
```

```
CREATE MEMORY STORE myStore;
CREATE OUTPUT WINDOW OWin1
    PRIMARY KEY (Col1)
    STORE myStore
    AS SELECT  A.Col1 col1, A.Col2 col2
    FROM IWin1 A;
```

Local Streams and Windows

Use a local stream, window, or keyed stream when the stream does not need an adapter, or to allow outside connections. Local streams, windows, and keyed streams are visible only within the containing CCL project, which allows for more optimizations by the CCL compiler. Streams and windows that do not have a qualifier are local.

Note

A local window cannot be debugged because it is not visible to the SAP HANA Streaming Run-Test tools such as viewer or debugger.

Example

This is a local stream:

```
CREATE SCHEMA mySchema (Col1 INTEGER, Col2 STRING);
CREATE LOCAL STREAM LStr1
    AS SELECT i.Col1 col1, i.Col2 col2
    FROM IStr1 i;
```

This is a local window:

```
CREATE SCHEMA mySchema (Col1 INTEGER, Col2 STRING);
CREATE MEMORY STORE myStore;
CREATE LOCAL WINDOW LWin1
    PRIMARY KEY (Col1)
    STORE myStore
    AS SELECT i.Col1 col1, i.Col2 col2
    FROM IStr1 i;
```

2.3.5 Making Windows Recoverable

By default, any window you create is assigned to a memory store. Memory stores hold all data in memory, and are not recoverable. You can make the contents of a window recoverable by assigning the window to a log store.

To start using log stores, you can change the default settings for all windows in the project, or you can assign log stores directly to specific windows. See Data Retention and Recovery with Stores [page 158] for more information.

2.4 Schemas

A schema defines the structure of data rows in a stream or window.

Every row in a stream or window has the same structure (schema), including the column names and datatypes and the order in which the columns appear. Multiple streams or windows can use the same schema, but each stream or window can only have one schema.

You can make a schema in one of the following ways:

- Create a named schema using the CREATE SCHEMA statement. Named schemas are useful when the same schema will be used in multiple places, since any number of streams and windows can reference a single named schema.
- Create an inline schema within a stream or window definition.

Example

Simple Schema CCL Example

This example uses a CREATE SCHEMA statement to create the named schema `TradeSchema`:

```
CREATE SCHEMA TradeSchema (
        Ts BIGDATETIME,
        Symbol STRING,
        Price MONEY(4),
        Volume INTEGER
);
```

This example uses a CREATE SCHEMA statement to make an inline schema:

```
CREATE STREAM trades SCHEMA (
        Ts bigdatetime,
        Symbol STRING,
        Price MONEY(4),
        Volume INTEGER
);
```

2.5 Adapters

Adapters connect SAP HANA smart data streaming to the external world.

An input adapter connects an input stream or window to a data source. It reads the data output by the source and modifies it for use in smart data streaming projects.

An output adapter connects an output stream or window to a data sink. It reads the data output by the smart data streaming project and modifies it for use by the consuming application.

Adapters are attached to streams and windows using the ATTACH ADAPTER statement. Start them by using the ADAPTER START statement. Some projects may need to start adapters in a particular order, for example, to load reference data before attaching to a live event stream. Adapters can be assigned to groups and the ADAPTER START statement can control the start-up sequence of the adapter groups.

See the *SAP HANA Smart Data Streaming: Adapters Guide* for detailed information about configuring individual adapters, datatype mapping, and schema discovery.

Related Information

[Adding an Input Adapter [page 306]](#)
[Adding an Adapter to a Project [page 75]](#)
[Adding an Output Adapter for SAP HANA [page 318]](#)
[SAP HANA Smart Data Streaming: Adapters Guide](#)

2.6 SAP HANA Reference Tables

Reference table queries help augment the streaming data in a SAP HANA smart data streaming project with information from a table or view in SAP HANA.

This CCL element enables you to establish a reference to a table or view in SAP HANA from within the smart data streaming project, then use the reference in a join along with streams and windows. When an event arrives via a stream or window, the reference executes a query on the table in the external database and uses the returned data in the join to enrich streaming data with information from the database.

To create a reference, you need:

- The name of the database service to use.
- The name of the table from which to retrieve information.
- The schema of the table.

You can also specify:

- The primary key of the table as the primary key of the reference.
- That the reference should attempt to reconnect when the connection is lost.
- How many attempts to make.
- How long to wait between attempts.

In CCL, use the CREATE REFERENCE statement to define the reference, then use the FROM and ON clauses to join data from the reference with streams and windows in the smart data streaming project. You can also use the **Reference** shape (found under Streams and Windows in the palette) within the SAP HANA Streaming Development perspective's visual editor.

In CCLScript, you can use an iterator over a reference from a local DECLARE block and in the Flex operator in the same way you use an iterator over a window. You may also iterate over a reference using the key search (if a primary key is defined), record-matching search, and for loop functionality. See the *SAP HANA Smart Data Streaming: CCL Reference* for more information on these statements and clauses.

Related Information

SAP HANA Smart Data Streaming: CCL Reference

2.7 CCL Functions

A function is a self-contained, reusable block of code that performs a specific task.

SAP HANA smart data streaming supports:

- Built-in functions, including aggregate, scalar, and other functions.
- User-defined CCLScript functions.

Built-in functions come with the software and include functions for common mathematical operations, aggregations, datatype conversions, and security.

Order of Evaluation of Operations

Operations in functions are evaluated from right to left. This is important when variables depend on another operation that must pass before a function can execute because it can cause unexpected results. For example:

```
integer a := 1;
integer b := 2;
max( a + b, ++a );
```

The built-in function `max()`, which returns the maximum value of a comma-separated list of values, returns 4 since ++a is evaluated first, so `max(4, 2)` is executed instead of `max(3, 2)`, which may have been expected.

2.8 Queries in CCL

CCL queries are attached to derived streams or windows to select data from one or more inputs and transform it into the desired output.

CCL embeds queries within CREATE STREAM and CREATE WINDOW statements in the same way that standard SQL uses CREATE VIEW statements. Unlike SQL, in CCL, SELECT is not a statement but rather is a clause used within a CREATE `<object-type>` statement.

Where the visual editor lets you select data using visual components referred to as simple queries, these queries are actually CCL statements that create a stream or window with an attached query.

To develop queries in CCL, refer to topics under CCL Query Construction [page 176], or see the *Statements* and *Clauses* topics in the *SAP HANA Smart Data Streaming: CCL Reference*.

Related Information

SAP HANA Smart Data Streaming: CCL Reference

2.9 Bindings between CCL Projects

Bindings enable data to flow between projects by allowing a stream, keyed stream, or window in one project to subscribe or publish to a stream, keyed stream, or window in another project.

A binding is a named connection from an input or output stream (or keyed stream or window) of one project to an input stream (or keyed stream or window) of another; you can configure it at either end.

- An input stream can subscribe to one or more streams in other projects. These streams do not have to be output streams—you can create an output binding on an input stream.
- An output stream can publish to one or more input streams in other projects. An output stream cannot receive incoming data, whether by subscription or publication.

Bindings reside in the project configuration file (`.ccr`), so you can change them at runtime. The streams being bound require compatible schemas.

See Using Bindings to Connect CCL Projects [page 173] for examples and more information.

3 Working with Projects in Web IDE

You can create and build CCL projects throughSAP Web IDE for SAP HANA.

With SP 12, SAP HANA smart data streaming introduces a CCL editing plugin to be used with SAP Web IDE for SAP HANA. SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) for building HANA applications. The SAP HANA smart data streaming plugin combined with SAP Web IDE for SAP HANA provide basic CCL editing capabilities, allowing users of the Web IDE to create and deploy simple CCL projects. The plugin is not a replacement for the SAP HANA studio, and does not provide the breadth of streaming design time tools that are available in the SAP HANA studio. You should continue to use SAP HANA studio as the primary development environment for SAP HANA Smart Data Streaming projects.

A streaming project, consisting of CCL and CCR files, is treated by Web IDE as a smart data streaming module within a multi-target application (MTA) project. An MTA project can contain multiple streaming modules. CCL and CCR files for a streaming module have to be located in a subdirectory of the MTA project root directory. You can edit smart data streaming projects by editing the corresponding smart data streaming module, using a text-based CCL editor. You can and compile and deploy smart data streaming projects on the SAP HANA system by using the build command within Web IDE. And, you can use the streaming run-time tool, which allows you to administer the smart data streaming projects that have been developed and deployed as modules in MTA projects, and to create users and assign privileges to these projects.

For more information about SAP Web IDE for SAP HANA, refer to the *SAP Web IDE for SAP HANA Reference* in the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.

1. Create a Multi-Target Application Project [page 35]
   Multi-Target Application (MTA) Projects are applications consisting of multiple sub-parts (called modules) which are developed using different programming paradigms and designed to run on different target runtime environments. In Web IDE, a smart data streaming project is one type of MTA module, so you'll need to create an MTA project before creating a streaming module.
2. Create a Smart Data Streaming Module [page 36]
   In SAP Web IDE for SAP HANA, a smart data streaming project (consisting of CCL and CCR files), is a module that you create within a Multi-Target Application (MTA) project.
3. CCL and CCR Files in the Streaming Module [page 37]
   Creating a streaming module automatically creates the `model` folder, with a CCL file (`model.ccl`) and a CCR file (`model.ccr`). You can modify or replace these files to customize the streaming module.
4. Build the Streaming Module [page 38]
   Build a streaming module automatically runs the smart data streaming project on the smart data streaming server, enabling you to upload data from a file, manually enter data, and view streams.
5. Streaming Run-Time Tool [page 39]
   The streaming run-time tool is an SAP Fiori Web-based application that allows you to easily manage projects that have been developed and deployed using SAP Web IDE for SAP HANA.

3.1 Create a Multi-Target Application Project

Multi-Target Application (MTA) Projects are applications consisting of multiple sub-parts (called modules) which are developed using different programming paradigms and designed to run on different target runtime environments. In Web IDE, a smart data streaming project is one type of MTA module, so you'll need to create an MTA project before creating a streaming module.

Prerequisites

- Ensure that XS Advanced (XSA), smart data streaming, and the streaming service broker are installed. For details, see *Ensuring that Smart Data Streaming Works with Web IDE* in the troubleshooting section of the *SAP HANA Smart Data Streaming: Installation Guide*.

  > **Note**
  >
  > Technically, you will be able to create an MTA project and a streaming module in Web IDE without having smart data streaming and the streaming service broker installed. However, you will not be able to build the streaming module, or access the streaming run-time tool.

- Ensure that you have the XSA application role collection `XS_CONTROLLER_USER`
- Ensure that you have the XSA controller role `SpaceDeveloper`. For details on XSA role collections and roles, see the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.

Procedure

1. In the SAP Web IDE for SAP HANA interface, right click the Web IDE workspace in which you will ultimately want to create the streaming module, and select **New** > **Project from Template**.
2. In the **Template Selection** pane, specify **Multi-Target Application Project**, and click **Next**.
3. In the **Basic Information** pane, enter a unique name for the MTA project, and click **Next**.
4. Click **Finish**.

   Creating the MTA project automatically generates the MTA descriptor (the `mta.yaml` file). The MTA descriptor contains a list of all entities, such as modules, resources, and properties that belong to an application or are used by it at run time, and the dependencies between them. For more information, see the *SAP Web IDE for SAP HANA Reference* in the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.

Task overview: Working with Projects in Web IDE [page 34]

Next task: Create a Smart Data Streaming Module [page 36]

3.2 Create a Smart Data Streaming Module

In SAP Web IDE for SAP HANA, a smart data streaming project (consisting of CCL and CCR files), is a module that you create within a Multi-Target Application (MTA) project.

Prerequisites

- Ensure that XS Advanced (XSA), smart data streaming, and the streaming service broker are installed. For details, see *Ensuring that Smart Data Streaming Works with Web IDE* in the troubleshooting section of the *SAP HANA Smart Data Streaming: Installation Guide*.

  > **Note**
  >
  > Technically, you will be able to create an MTA project and a streaming module in Web IDE without having smart data streaming and the streaming service broker installed. However, you will not be able to build the streaming module, or access the streaming run-time tool.

- Ensure that you have the XSA application role collection `XS_CONTROLLER_USER`
- Ensure that you have the XSA controller role `SpaceDeveloper`. For details on XSA role collections and roles, see the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.
- You have created an MTA project.

Context

An MTA project can contain multiple smart data streaming modules.

Procedure

1. In the SAP Web IDE for SAP HANA interface, right click the folder representing the MTA project in which you want to create the streaming module, and select **New** > **Smart Data Streaming Module**.
2. In the **Basic Information** pane, enter a unique name for the streaming module, and click **Next**.
3. Click **Finish**.

   Creating the streaming module automatically creates the `model` folder, with a CCL file (`model.ccl`) and a CCR file (`model.ccr`).

   Each time you add a module to the MTA project, the MTA Descriptor (the `mta.yaml` file) is automatically extended with the default information for this module. For more information, refer to the *SAP Web IDE for SAP HANA Reference* in the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.

Task overview: Working with Projects in Web IDE [page 34]

Previous task: Create a Multi-Target Application Project [page 35]

Next: CCL and CCR Files in the Streaming Module [page 37]

3.3 CCL and CCR Files in the Streaming Module

Creating a streaming module automatically creates the `model` folder, with a CCL file (`model.ccl`) and a CCR file (`model.ccr`). You can modify or replace these files to customize the streaming module.

Use Continuous Computation Language (CCL) to specify information about the streams, and other elements that characterize the streaming module in the CCL file, `model.ccl`. Use this file to define the smart data streaming project (known as the streaming module in the Web IDE environment). For the syntax of specific CCL statements, clauses and functions, see the *SAP HANA Smart Data Streaming: CCL Reference*.

The streaming module also has a CCR (`model.ccr`) file. The CCR file contains runtime and deployment configuration, including adapter, parameter, and binding definitions.

CCL and CCR files for a streaming module must reside in the module's `model` folder.

You can:

- Remove or rename a CCL or CCR file by right-clicking the file and selecting **Delete** or **Rename**.
- Add a new CCL or CCR file by right-clicking the `model` folder and selecting **New** > **File**.
- Import an existing CCL or CCR file by right-clicking the `model` folder and selecting **Import** > **From File System**.
- Make a local copy of a streaming module developed in Web IDE by right-clicking the module and selecting **Export**.

> **Note**
>
> You cannot import an existing smart data streaming project directly into an MTA project. However, Web IDE supports the cloning of source code from a git repository. If you maintain your smart data streaming project source code in the git repository, you can obtain the entire smart data streaming project by right-clicking the MTA project's streaming module and selecting **Git** > **Clone Repository**.

Parent topic: Working with Projects in Web IDE [page 34]

Previous task: Create a Smart Data Streaming Module [page 36]

Next task: Build the Streaming Module [page 38]

3.4 Build the Streaming Module

Build a streaming module automatically runs the smart data streaming project on the smart data streaming server, enabling you to upload data from a file, manually enter data, and view streams.

Prerequisites

- Ensure that XS Advanced (XSA), smart data streaming, and the streaming service broker are installed. For details, see *Ensuring that Smart Data Streaming Works with Web IDE* in the troubleshooting section of the *SAP HANA Smart Data Streaming: Installation Guide*.

  > **Note**
  >
  > Technically, you will be able to create an MTA project and a streaming module in Web IDE without having smart data streaming and the streaming service broker installed. However, you will not be able to build the streaming module, or access the streaming run-time tool.

- Ensure that you have the XSA application role collection `XS_CONTROLLER_USER`
- Ensure that you have the XSA controller role `SpaceDeveloper`. For details on XSA role collections and roles, see the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.
- You have created an MTA project, and a smart data streaming module.

Procedure

1. In the SAP Web IDE for SAP HANA interface, right click the smart data streaming module and select **Build**.
2. Monitor the build status in the build log.

   When the build is complete, the project is running. You can view all running and stopped projects using the Streaming Run-Time Tool [page 39].

Task overview: Working with Projects in Web IDE [page 34]

Previous: CCL and CCR Files in the Streaming Module [page 37]

Next: Streaming Run-Time Tool [page 39]

3.5 Streaming Run-Time Tool

The streaming run-time tool is an SAP Fiori Web-based application that allows you to easily manage projects that have been developed and deployed using SAP Web IDE for SAP HANA.

Similar to some of the streaming tiles in the SAP HANA cockpit, but specifically used in conjunction with Web IDE, the streaming run-time tool is available when XS advanced and smart data streaming have been installed on your SAP HANA system. For details, see *Ensuring that Smart Data Streaming Works with Web IDE* in the troubleshooting section of the *SAP HANA Smart Data Streaming: Installation Guide*.

Access through Web Interface

You can access the streaming run-time tool through a web interface using your XS Advanced credentials. To locate the URL for this interface,

1. Log on to a host in the SAP HANA system as the `<sid>`adm user.
2. Run the `xs login` command and provide XSA credentials.
3. Run the command `xs app sds-rtt-ui`.

Alternatively, you can log on to the XS Advanced Administration and Monitoring tool, and select the Application Monitor tile.

Security

Smart data streaming has defined two XSA application security scopes for the streaming run-time tool: `SDSProjectAdmin` and `SDSUserAdmin`. Based on these scopes, there are three role templates:

Role Template	Description
`xsa_sds_rtt_viewer_template`	A user with this role does not have the `SDSProjectAdmin` or `SDSUserAdmin` scope, and can view the status of smart data streaming projects with the **Projects** tile of the streaming run-time tool, but can't start or stop a project. The **User Management** tile is not visible.
`xsa_sds_rtt_developer_template`	A user with this role has the scope `SDSProjectAdmin`, and can view, start or stop smart data streaming projects with the **Projects** tile of the streaming run-time tool. The **User Management** tile is not visible.
`xsa_sds_rtt_administrator_template`	A user with this role has both the `SDSProjectAdmin` and `SDSUserAdmin` scopes. They can view, start or stop smart data streaming projects with the **Projects** tile of the streaming run-time tool. They can add, delete and manage users with the **User Management** tile.

To assign these to XSA users, you first need to create custom role collections: the `xsa_sds_developer` role collection to include the developer role and the `xsa_sds_admin` role collection to include the admin role. This can be done using the XSA admin tool. After creating the role collections, you can assign them to users by using **SAP HANA Web-based Development Workbench: Security** (which can be opened through the SAP HANA cockpit **User and Role Management** tile). For more information about XSA scope, application role and role collection, refer to the *SAP HANA Developer Guide (for SAP HANA XS Advanced Model)*.

In this section:

Parent topic: Working with Projects in Web IDE [page 34]

Previous task: Build the Streaming Module [page 38]

3.5.1 Managing a Project

With the streaming run-time tool you can view, start and stop smart data streaming projects that have been developed and deployed as modules in MTA projects using SAP Web IDE for SAP HANA.

Prerequisites

- To view project status, you need one of three XSA roles: `xs_sds_rtt_viewer`, `xsa_sds_rtt_developer` or `xsa_sds_rtt_adminstrator`
- To start or stop projects, you need either the `xsa_sds_rtt_developer` or `xsa_sds_rtt_adminstrator` XSA roles

Context

In smart data streaming, projects are deployed in workspaces on a streaming cluster. A workspace provides a namespace for the project and also allows you to control permissions at the workspace level. With the streaming run-time tool, when you select an instance of the streaming service broker, you have access to all running and stopped projects in a specific workspace.

Procedure

1. Select the **Project Tool** tile from the streaming run-time tool home page.
2. Click the list icon and select a streaming service broker instance.
3. Select a smart data streaming project from the list.

   The project details show the workspace name, the project name and the node name, as well as whether the project is running or stopped. If the project is running, each stream is also displayed.
4. Select **Start** or **Stop** to change the status of the project.

3.5.2 Managing Users

With the streaming run time tools you can add a user in order to assign project privileges, delete a user, or assign project privileges to existing users.

Prerequisites

To view and use the **User Management** tile , you need to have the `xsa_sds_rtt_adminstrator` XSA role.

Context

In smart data streaming, projects are deployed in workspaces on a streaming cluster. A workspace provides a namespace for the project and also allows you to control permissions at the workspace level. With the streaming run-time tool, when you select an instance of the streaming service broker, you have access to all running and stopped projects in a specific workspace. When you create a user, that user is restricted to the selected streaming workspace.

Procedure

1. Select the **User Management** tile from the streaming run-time tool home page.
2. Click the list icon and select a streaming service broker instance.
3. To add a user, click .
   a. Check or uncheck each of the privileges for the new user.
   b. Click **Add**. A new User Name and Password are auto-generated. The new user is restricted to the streaming service broker instance you selected.
   c. In the dialog box, take note of the User Name, and unmask the password by clicking .

4. To delete a user, select the user from the list.
    a. Click **Delete**.
    b. In the dialog box, click **OK**.
5. To modify the privileges of an existing user, select the user from the list.
    a. Click **Edit**.
    b. Check or uncheck each of the privileges for that user.
    c. Click **Save**.

4 Working with Projects in the Streaming Plugin for SAP HANA Studio

The streaming plugin for the SAP HANA studio provides two specific perspectives to create, edit, test and deploy CCL projects: the SAP HANA Streaming Run-Test and SAP HANA Streaming Development perspectives.

You can access all smart data streaming components and features from within SAP HANA studio.

The perspectives provide an environment for working with sample streaming projects, and for running and testing applications with various debugging tools. You can record and playback project activity, upload data from files, manually create input records, and run ad hoc queries against the server.

You can compile projects into an executable project file. The project file can be shared, tested, developed, and deployed on any operating system with the SAP HANA smart data streaming plugin for SAP HANA studio installed.

Data-Flow Programming

In data-flow programming, you define a set of event streams and the connections between them, and apply operations to the data as it flows from source to output.

Data-flow programming breaks a potentially complex computation into a sequence of operations with data flowing from one operation to the next. This technique also provides scalability and potential parallelization, since each operation is event-driven and independently applied. Each operation processes an event only when it is received from another operation. No other coordination is needed between operations.

The sample project shown in this figure is a simple example of data-flow programming. Each of the continuous queries in this example—the VWAP aggregate, the IndividualPositions join object, and the ValueByBook aggregate—is a type of derived stream, as its schema is derived from other inputs in the diagram, rather than originating directly from external sources.

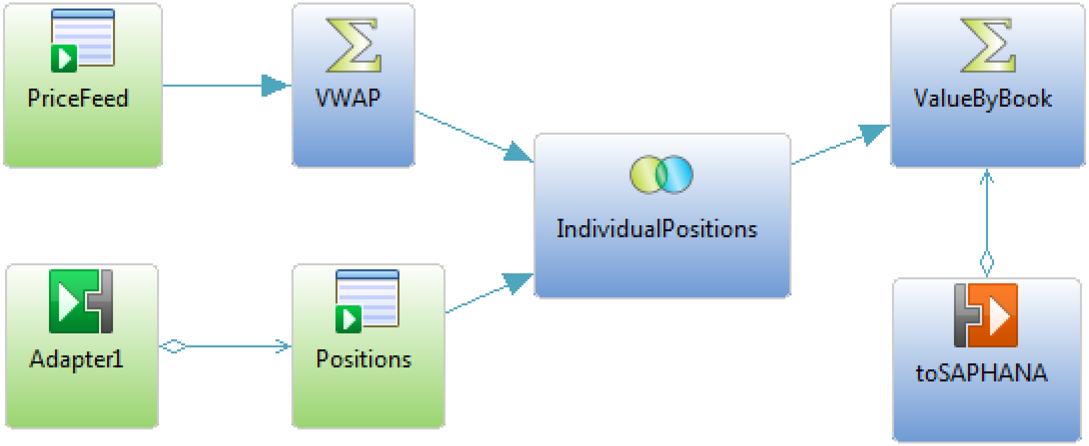
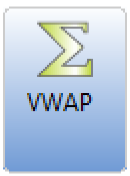
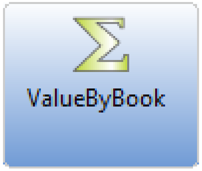


Figure 4: Data-Flow Programming – Simple Example

You can create derived streams in a diagram using the simple query elements provided in the studio visual editor, or by defining your own explicitly.

Element	Description
PriceFeed 	Represents an input window, where incoming data from an external source complies with a schema consisting of five columns, similar to a database table with columns. In smart data streaming, however, the streaming data is not stored in a database.
Positions 	Another input window, with data from a different external source. Both Positions and PriceFeed are included as windows, rather than streams, so that the data can be ag-gregated.
VWAP 	Represents a simple continuous query that performs an aggregation, similar to a SQL SELECT statement with a GROUP BY clause.

Element	Description
IndividualPositions  	Represents a simple continuous query that performs a join of Positions and VWAP, similar to a SQL FROM clause that produces a join.
ValueByBook  	Another simple query that aggregates data from the stream IndividualPositions.

In this section:

4.1 Streaming Basics in SAP HANA Studio

The smart data streaming plugin for SAP HANA studio is an interactive development environment for building and testing event-based applications. It includes the SAP HANA Streaming Development and SAP HANA Streaming Run-Test perspectives, as well as a visual and CCL editor.

Perspectives

To access one of the smart data streaming perspectives, select **Window** > **Open Perspective** > **Other**, then open the desired perspective. The following table describes the activities you can perform in the different perspectives.

Perspective	Activities
SAP HANA Streaming Development	• Create and edit projects. • Develop projects and diagrams in the visual editor, a graphical editing environment. • Develop projects in the CCL editor, a text-oriented editing environment where you edit CCL code. • Compile projects.
SAP HANA Streaming Run-Test	• Start and connect to servers. • Run projects. • Enter test data by uploading data files to a server, or entering data manually to a stream. • Publish data. • Execute a query against a running project. • Use the event tracer and debugger to set breakpoints and watchpoints, and trace the flow of data through a project. • Record incoming event data to a playback file, and play back captured data into a running project. • Monitor performance.

Streaming Development Perspective Components

The visual editor, CCL editor, and other tools and views in the SAP HANA Streaming Development perspective allow you to create, view, and edit a diagram or CCL file. When viewing the SAP HANA Streaming Development perspective, its components are:

Editor	Streaming Development perspective where you edit the diagram (in the visual editor) or CCL (in the CCL editor). The visual and CCL text editors are completely integrated. When you save and switch to the other editor, your work is saved there as well. You can switch between editors in several ways, including right-clicking and selecting **Switch**, or choosing **F6**.
Palette	Includes groups of tools used to create new CCL elements on the diagram. Most shapes in the palette correspond to a CCL statement.

Project Explorer	Provides a hierarchical tree structure of folders and files. Adding a new smart data streaming project displays it in the project explorer. As you add new elements to the project, such as new windows or streams, you can display them in the project explorer by selecting **Refresh** from the project explorer context menu. The **Toggle Streaming Nature** option in the context menu toggles the appearance of smart data streaming-specific context menu options for individual projects in the project explorer.
Properties view	Displays the properties of the object selected in the diagram. You can also set properties in this view, and edit expressions.
Outline view	Provides an index to all elements in the diagram as a hierarchical tree structure. Also shows the order in which adapters are started. Right-click an element in this view to show it in the diagram, delete it, modify it, or add a child element.
Overview	Helps you understand the big picture, and navigate easily to different areas of a large, complex diagram. For large diagrams you can scroll the editor by dragging the gray box in the overview.
Search	Provides full-text search capability for finding text strings in the workspace. Useful in navigating the file explorer, and project contents in the CCL editor. You can filter search results, and copy, remove, or replace results found.
Problems	Displays errors found when you compile a project.
Console	Displays messages generated when interacting with smart data streaming components.

Streaming Run-Test Perspective Components

You can access various tools to test, monitor, debug, and fine-tune a project from the SAP HANA Streaming Run-Test perspective. The components of the SAP HANA Streaming Run-Test perspective are:

Server View	Start and connect to available servers. Your first project is there, already running.
Manual Input view	Manually create and publish events as input to a stream or window.
Playback view	Record data flowing into a running project, or play back recorded files.
File Upload view	Publish an existing data file to an input stream or window.
SQL Query view	Run a snapshot SQL query. It captures a snapshot of the current window state and displays results in the Console.
Console view	Review log messages and other tracing and debugging information useful to developers.
Stream view	Show the events of an output stream or the retained events in an output window of a running project.
Monitor view	Monitor performance of a running project.
Debugger view	Debug a project by setting breakpoints and watchpoints.
Event Tracer view	Trace the flow of data through a project.

Eclipse Terms

The smart data streaming plugin for SAP HANA studio uses some Eclipse terms:

Term	Description
View	A named window in the studio user interface. Views are tabbed so several can overlap and occupy the same pane. The SAP HANA Streaming Run-Test perspective, for example, includes the server view, manual input view, file upload view, and playback view. You can move, minimize, and resize views.
Perspective	A named set of views. Like views, perspectives are tabbed so several can use the same space. Perspectives in studio include SAP HANA Streaming Development and SAP HANA Streaming Run-Test.
Workspace	The studio workspace is a directory that stores projects. This workspace is distinct from cluster workspaces, which are logical groupings of deployed projects.

In this section:

4.1.1 Accessing Smart Data Streaming in SAP HANA Studio

Access SAP HANA smart data streaming functionality within the SAP HANA studio by opening the SAP HANA Streaming Run-Test and SAP HANA Streaming Development perspectives.

Procedure

In SAP HANA studio, go to **Window** > **Open Perspective** > **Other**, then open the **SAP HANA Streaming Development** or **SAP HANA Streaming Run-Test** perspective.

For more information on each perspective, see Streaming Basics in SAP HANA Studio [page 46].

4.1.2 Editing in the CCL Editor

The CCL editor is a text authoring environment within SAP HANA Streaming Development perspective for editing CCL code.

Context

You can work in the CCL editor exclusively, or use it as a supplement to the visual editor. The CCL editor offers syntax completion options, syntax checking, and error validation.

A single CCL file can be open in only one editor at a time. The visual and CCL editors are completely integrated: when you save and switch to the other editor, your work is saved there as well.

You may find that you prefer the CCL editor when adding advanced features to your projects.For example, you can add:

- Complex queries that exceed the capabilities of the visual editor.
- DECLARE blocks for declaring project variables, parameters, datatypes, and functions.
- CCLScript event handlers that you invoke with Flex operators.
- Reusable modules and schemas that can be used multiple times in a project, or across projects.

Several features simplify the process of editing CCL code in the CCL editor:

Feature	Description
Completion Proposals	Activate completion proposals in workspace [**Ctrl + Space**].
Case-Insensitive Syntax Highlighting	Done automatically when editing CCL code.
Error Validation/Syntax Checking	Access the Problems view to see errors in CCL code.
Compile and Report Compilation Errors	Access the Problems view to see errors in CCL code.

For CCL language details, see the *SAP HANA Smart Data Streaming: CCL Reference*.

Procedure

1. Select the SAP HANA Streaming Development perspective.
2. In the project explorer, expand the project container, and double-click the `.ccl` file name to open it in the CCL editor.

   > **Note**
   >
   > Advanced CCL users can include multiple CCL files in the same project by using an IMPORT statement to import shared schemas and module definitions from another file. When using the IMPORT statement to import one CCL file into another, both CCL files must be contained within the same project. If not, you receive an error message stating that the file cannot be located.

3. Begin editing text in the CCL editor window.

   > **Tip**
   >
   > If you open a `.ccl` file in the CCL editor when the same project is open in the visual editor, the CCL editor opens in read-only mode and you cannot edit the file.
   >
   > Close both the visual editor and CCL editor for the project, and then reopen the project in the CCL editor.

   > **Note**
   >
   > Backslashes within string literals are used as escape characters; specify any Windows directory paths with two backslashes.

4. (Optional) Press **Ctrl+Space** to show a syntax completion proposal.
5. (Optional) To insert CREATE statement template code, right-click, choose **Create**, and then choose the element to create.
6. (Optional) To search for text, choose **Search** > **File**.
   You can also start a new search from the link in the **Search** view, when no search results are visible.

   > **Tip**
   >
   > Double-click a search result to highlight it in the CCL editor.

7. (Optional) To write comments in CCL that appear as tooltips for shapes in the visual editor, insert a comment immediately preceding the declaration statement for the corresponding shape.

   For example, to insert a tooltip for an input window, type:

   ```
   /**InputWindowInStudio*/
   CREATE INPUT WINDOW InputWindow1 ;
   ```

8. Choose **File** > **Save** (**Ctrl+S**) to save the `.ccl` file and the project.

Related Information

SAP HANA Smart Data Streaming: CCL Reference

4.1.3 Editing in the Visual Editor

In the visual editor, the project is represented by one or more diagrams that show streams, windows, adapters, and the data flows between them. Add or remove shapes and connections or edit shape attributes to make changes to a project.

Context

The visual editor lets you create and edit projects while visualizing the data flow and navigating within the project. The visual and CCL editors are completely integrated. When you save and switch to the other editor, your work is saved there as well.

Diagrams

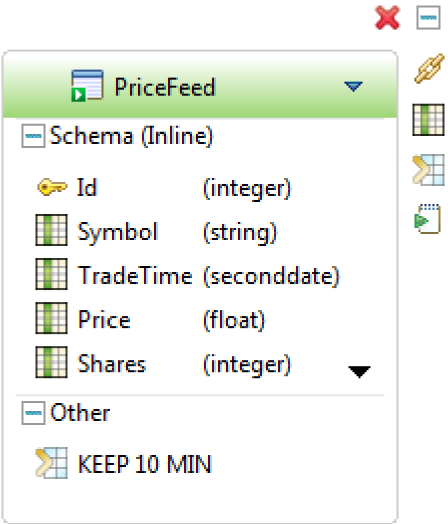
When you open a project in the visual editor, the project shows a collection of stream and window shapes that are connected with arrows showing the flow of data. You develop the project by selecting new input and output streams, windows, and other elements from the palette, dropping them onto the diagram, connecting them, and configuring their behavior.

Every project has at least one diagram. A diagram in the visual editor is a projection of the associated CCL statements in the project.

When you add a shape or other element to a diagram, it is automatically added to the project when you save. You can delete an element from a diagram only, or from the project.

Display diagrams in verbose or iconic mode:

iconic	compartments are collapsed to save space. 
verbose	all compartments in elements are visible.



- To expand or collapse all shapes in the diagram, use the **All Verbose** or **All Iconic** buttons on the main toolbar.
  - To expand an individual shape, select it and click the "+" box in the shape toolbar.
  - To collapse an individual shape, select it and click the "-" box in the shape toolbar.
- To change the orientation, in the visual editor toolbar click **Layout left to right** or **Layout top down**.

For more display options, right-click an object or the diagram surface and choose from the context menu.

Shapes

Each shape in the palette creates a specific type of stream or window, adapter, connection, reusable schema or module, or a store, to create a data flow.

Shape	Purpose	Usage
Connector	Creates flows between streams and windows, establishes references between streams and shared components, or attaches notes to shapes.	Click to select the connector tool, then click each of the shapes in the diagram to be connected..
Note	Creates a comment on the diagram only. This comment does not appear in the CCL file.	Documents additional user generated information in the SAP HANA Streaming Development perspective for a project in studio.
Input Stream	The entry point for unkeyed event streams into a project. Receives data from either an input adapter or an external publisher.	A stream does not retain any data and does not have a state. Data in an input stream is not keyed.

Shape	Purpose	Usage
Derived Stream (Local) Derived Stream (Output)	Applies a continuous query to data arriving from another stream or window to produce a new stream.	Streams do not retain data and do not have keys. They are "insert only," meaning that their output consists only of inserts. Input must be a stream or a stream-window join. By default, new streams (including derived streams) are output, so they are visible to external subscribers. You can change the property to local.
Derived Window (Local) Derived Window (Output)	Applies a continuous query to data arriving from another stream or window. Retains data, and retention rules can be set.	Data must be keyed so that every row has a unique key. Processes inserts, updates, and deletes both as local and output. You can use the toolbar to change the output to local, if you do not want it visible to external subscribers.
Input Window	The entry point for event streams into a project where incoming events have primary keys and there is a desire to maintain a window of event data. Supports opcodes (insert, update, delete, upsert). Use this as an entry point for event streams if: • The stream contains insert, update and delete events, or, • You need to retain a set of incoming events.	Window size can be set by row count with a fixed number of input records, or by time with records being kept for a specified period. The window must be keyed, that is, every row must have a unique key value.
Flex	A programmable operator that uses custom CCLScript scripts to process incoming events.	A Flex operator can take input from any number of streams and/or windows and will produce a new derived stream or window (either local or output).
Aggregate	Takes input from a single stream or window and groups records using a common attribute. Produces a single output record for each group. Uses aggregate functions like sum(), count(), and so on.	Always creates a new window. Requires a GROUP BY element. You can optionally set window size using retention rules.
Compute	Takes input from a single source and computes a new record for every record received. Allows you to change the schema on events, computing new fields and changing existing fields.	Produces a derived stream when the input is a stream. Produces a derived keyed stream when the input is a keyed stream. Produces a derived window when the input is a window.
Filter	Takes input from a single source and applies a filter. Creates a stream of records that match the filter criteria.	Produces a derived stream when the input is a stream. Produces a derived keyed stream when the input is a keyed stream. Produces a derived window when the input is a window.

Shape	Purpose	Usage
Join	Takes input from two or more sources and joins them based on common data elements.	See Joins [page 183].
Pattern	Takes input from two or more sources and detects patterns of events. One output record is produced every time a pattern is detected.	Detects patterns in the incoming data. See Pattern Matching [page 191].
Union	Merges input from two or more sources. One output record is produced for every input record.	All inputs must have a common schema.
Named Schema	Represents a CCL CREATE SCHEMA statement. Reusable definition of column structure that can be referenced by streams and windows.	A schema defined outside of an object that can be used in multiple places, since any number of streams and windows can reference a single named schema.
Module	Represents a CCL CREATE MODULE statement. Creates a new module that can be used in one or more places in the project.	A module can contain all the same elements as a project and provides for reuse.
Log Store	Stores data held in windows. Provides disk-based recovery but is slower than a memory store.	By default, new windows are assigned to a memory store. Where recoverability of data in a window is required, create a log store and assign the window to it.
Memory Store	Stores data held in windows.	Faster than a log store but does not recover data after shutdown: • (Default) Created implicitly by the CCL compiler, if no other store is specified. • (Optional) Created explicitly, with windows assigned to specific stores, to optimize performance.
Input Adapters	Connects an input stream or input window to an external data source.	Must be connected to either an input stream or input window. To use schema discovery—that is, to import the schema from the source—add the input adapter first, and then use schema discovery to create a connected input stream or window with the imported schema.
Output Adapters	Connects an output stream or window to a destination.	Must be connected to either an output stream or an output window.

Shape	Purpose	Usage
Reference	A CCL element that establishes a reference from an external database table to a project in smart data streaming. Use reference—in joins and in CCLScript programs—inside a module as well as within the main body of your project.	Can be used for data lookup and to enrich data streaming in SAP HANA smart data streaming with information permanently stored in an external database table. For example, customer address and credit card information. For schema discovery, datatypes in the source schema for the reference must be compatible with those found in the external database table the reference queries. If incompatible, unexpected behavior such as an abnormal shutdown can occur.

Procedure

1. In the SAP HANA Streaming Development perspective, navigate to the project explorer.
2. Double-click the `.cclnotation` file name of your project to open it in the visual editor.

   Note

   If you see an error message indicating that the linked file does not exist, this is likely because the name of the `.ccl` linked to the `.cclnotation` file has changed. To open the `.cclnotation` file, right-click and choose **Change Linked CCL File**, then select the `.ccl` file to be used with this `.cclnotation` file.
3. Click in the diagram to begin editing using the palette.

   Tip

   To make the visual editor window full-screen, double-click the `<name>`**:Diagram** tab at the top. Double-click again to revert.
4. Create streams, windows, and shared components, relate them using continuous queries, and attach them to adapters by clicking a shape tool in the palette and then on an empty area in the diagram.

   Tip

   To view actions needed to complete a shape definition, hover the mouse over the shape in the diagram.

   The border changes from red to grey when the shape definition is complete and correct.
5. Save as you go (**Ctrl+S**).
   This saves changes to both the `.cclnotation` file (the diagram) and the `.ccl` file (the CCL).
6. (Optional) To add a comment to a shape that will appear within a tooltip when you hover, select the comment field in the Properties view for the highlighted shape.

Tip

Enable tooltip comments. Select the **Show comments in shape tooltips** checkbox in **Window > Preferences > SAP HANA smart data streaming > CCL Editor**.

7. To toggle between the visual editor and the CCL editor, choose **Switch to Text** or **Switch to Visual** (**F4**).
8. To close the diagram, press **Ctrl+W** or **Ctrl+F4**, or click the **X** on the tab at the top of the editor .

Note

The visual editor, like other graphical user interfaces, offers several ways to accomplish most tasks, although this guide may not list all of them. For example, in many contexts you can carry out an action by:
- Clicking a button or other icon in a shape, or on the main toolbar.
- Using a shortcut key.
- Double-clicking an element to open it.
- Right-clicking to select from the context menu.
- Selecting from the main menu bar.
- Editing element values in the Properties view.

SAP HANA studio also includes features common to Eclipse-based applications.

4.1.4 Project Execution and Testing

Run and test all aspects of a streaming project using the SAP HANA Streaming Run-Test perspective.

During development, you can use the SAP HANA Streaming Run-Test perspective to run any compiled project against a server, view data flowing through the streams and windows defined in the project, execute queries, and use debugging tools. Your project configuration and licensing determine the type of server connections you can use when running projects. Some adapters also have special licensing requirements.

In the studio you can connect immediately to a cluster to run projects, using default security established for smart data streaming during installation. A cluster consists of a group of server nodes, which are processes that run on hosts. A cluster can have a single node or multiple nodes.

In a production environment, you typically run projects on a server. Administrators monitor, manage, and configure smart data streaming server nodes, clusters, and projects using SAP HANA cockpit. Command-line utilities and procedures are also available, as discussed in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

Related Information

Testing the Project with Recorded Data [page 324]
SAP HANA Smart Data Streaming: Configuration and Administration Guide

4.1.5 SAP HANA Navigator

Use the SAP HANA Navigator to select tables and views within SAP HANA, then drop them directly into a project diagram in the studio visual editor. This creates a new adapter reference or named schema in the streaming project.

You can then use the new reference in a join along with SAP HANA smart data streaming streams and windows. When an event arrives via a stream or window, the reference executes a query on the table in SAP HANA and uses the returned data in the join to enrich streaming data with information from the database.

4.1.6 The Studio Log File

The SAP HANA studio logs activity and records it in a log file. Access this log file to view studio activity and to help troubleshoot events such as unexpected shut down.

The studio log file resides in your workspace directory under `workspace/.metadata/.log`. View the log within studio:

1. Select **Help > About Studio**.
2. Click **Configuration Details**.
3. Click **Configuration**.
4. Click **View Error Log**.
5. If prompted, select a text editor to view the file with.

The log provides read-only details on internal studio activity. You cannot modify the file to change what it reports on, or its level of verbosity.

4.1.7 Customizing the Streaming Work Environment

Customize your smart data streaming plugin for SAP HANA studio interface to work the way you prefer.

Context

> i Note
>
> As an Eclipse-based application, the smart data streaming plugin for SAP HANA studio automatically includes many features not specific to SAP HANA smart data streaming. Features documented here have been tested with the smart data streaming plugin for SAP HANA studio. Other Eclipse features may not work as expected. For example, the Team Synchronizing perspective is not supported.

In this section:

Editing Preferences for Smart Data Streaming [page 58]

Use the **Preferences** dialog to customize your Streaming plugin behavior in SAP HANA studio.

Keyboard Shortcuts [page 59]
Use keyboard shortcuts to quickly access various functions within the visual or CCL editors.

Manual Input Settings [page 59]
Set default values and formats for datatypes of the data you publish to a stream from the **Manual Input** view. This view is helpful for simplifying output for testing purposes as you prepare to create a project in studio.

Setting CCL Compiler Options [page 61]
You can adjust the CCL compiler options.

4.1.7.1 Editing Preferences for Smart Data Streaming

Use the **Preferences** dialog to customize your Streaming plugin behavior in SAP HANA studio.

Context

You can also access many of these preferences from the related view in the Streaming plugin for SAP HANA studio.

Procedure

1. Choose **Window** > **Preferences** .
2. Select and expand **SAP HANA smart data streaming** to find the preferences you want to set. All preference settings are optional.

SAP HANA smart data streaming	○ Set which editor you want to open when you create a new project. ○ Set defaults for the server URL, the workspace, and HANA Service Entry. ○ Set an operation time limit for the studio to time out when starting, stopping, and compiling projects (default is 5 minutes). If communication between the studio and the Java SDK times out, the following error message appears in the **Problems** view: `failed to create input stream: Read timed out`. If this occurs, then manually restart the operation.
CCL Text Editor	○ Set various defaults for refactoring, syntax coloring, and templates for statements and schemas. ○ Set whether to have the Text Editor automatically close punctuation and comments.
CCL Visual Editor	○ Use the **Shape Colors** tab to choose defaults for creating and displaying shapes in diagrams. ○ Use the **Shape Fonts** tab to set the font used in shapes.

Run Test	○ Use the **Run Test** tab to set defaults for the server workplace and connections, add new connections, set limits and filters for the stream view and server view, and set other options for running projects in studio. ○ Use the **Data Input Settings** tab to set file upload and SQL Query view options. ○ Use the **Security Settings** tab to configure Kerberos and RSA authentication for when projects are run and deployed.

3. On each preference dialog, either:
    - Click **Apply** to save the new settings, or,
    - Click **Restore Defaults** to revert any changes you make.

    Only the settings in the current dialog are applied or restored.

    > **Note**
    >
    > You may be asked to restart SAP HANA studio for your changes to take effect.

4. Click **OK** to exit the Preferences dialog.

4.1.7.2 Keyboard Shortcuts

Use keyboard shortcuts to quickly access various functions within the visual or CCL editors.

To show a list of all available keyboard shortcuts, press **Ctrl+Shift+L**.

To add a keyboard shortcut:

1. Select **Window** > **Preferences** > **General** > **Keys**.
2. Select a command from the list.
3. In the **Binding** field, add the new shortcut.
4. Click **OK**.

4.1.7.3 Manual Input Settings

Set default values and formats for datatypes of the data you publish to a stream from the **Manual Input** view. This view is helpful for simplifying output for testing purposes as you prepare to create a project in studio.

To configure these settings, choose **Window** > **Preferences**, then expand **SAP HANA smart data streaming** and **Run Test**. Settings for most datatypes are in **Manual Input Settings** preferences. Settings for the money(n) datatype are in **Manual Input Settings - Money Types** preferences.

> **Note**
>
> When you double-click an input field containing a negative value, only the text up to the minus symbol is selected. If you want to select the value including the minus symbol, single-click and highlight the text. This affects the **Manual Input Settings** and **Manual Input Settings - Money Types** preferences, and the **Manual Input** view in studio.

Setting	Description
Publish Multiple Rows	Indicates whether data from an input stream is published in single instances or as multiple rows.
Use Current Date	Indicates whether data should be published under the current date or maintain its historical date.
Interpret Date values in Manual Input and Stream Viewer as UTC	Indicates whether Manual Input date values are interpreted as UTC or in the local time zone. **Note** This has no effect on the Playback tool.
binary	Indicates a binary value to be published to a stream. Use this setting to monitor the binary value of a stream by placing a traceable value in the field.
boolean	May be set to **True** or **False**.
string	Indicates the default value studio accepts for string types.
integer	Indicates the default value studio accepts for integer types. Does not accept values with decimal points.
float	Indicates the default value studio accepts for float types.
long	Indicates the default value studio accepts for long types.
interval	Indicates the default value studio accepts for interval types.
seconddate	Indicates the default value for seconddate types. Click **Select** to open a calendar dialog and choose a default date with second precision.
bigdatetime	Indicates the default value for bigdatetime types. Click **Select** to open a calendar dialog and choose a default bigdatetime with microsecond precision.
msdate	Indicates the default value for msdates types. Click **Select** to open a calendar dialog and choose a default timestamp with millisecond precision.
time	Indicates the default value studio accepts for time types.

Note

You see an error message at the top of the preference window when you enter incorrect characters, or exceed the number of allowed characters in the field.

Related Information

Manually Entering Data to a Stream [page 114]

4.1.7.4 Setting CCL Compiler Options

You can adjust the CCL compiler options.

Procedure

1. Choose **Window** > **Preferences**.
2. Click **SAP HANA smart data streaming** > **Run Test**, and go to **Compiler output directory**.
3. To change the directory for your compiled projects, click **Change...**, select a directory, and click **OK**.
4. To confirm any other changes, click **OK**.

> **Note**
>
> By default, the compile directory is set to `bin`, which means the `.ccx` files are created in a subdirectory relative to the project's directory.

4.1.8 Sample Streaming Projects for Studio

Smart data streaming in SAP HANA studio includes several completed example projects.

Customize the Welcome page in SAP HANA studio to show samples of completed streaming projects. Click the Customize Page icon in the top right corner of the Welcome page to open the **Customize** dialog. Select the **Home** tab and in the **Root Pages** section, select the **Samples** option and click **OK**. The Welcome page now displays a Samples subpage.

You can view the examples in studio and run them against sample data installed with the product. Stepping through examples is an ideal way to watch a simplified set of event data flow through the system.

The examples include:

Indexes Calculation	Shows how continuous computations can be applied to a stream of market prices to deliver insight into the market. This example demonstrates reusable modules. Each of the market calculations is defined in an external module, that is, a module defined in a separate CCL file, and then imported into the project. Parameters (in this case, time and intervals) are set when the module is called.
Pattern Matching	Provides a simple example of situation detection: watching for a pattern of events. The example demonstrates how to watch for employee fraud in a retail setting, based on transaction patterns from a point-of-sale system. The example applies three filter queries to an input stream of transactions, and then uses a pattern query (CCL MATCHING clause) to produce a possible fraud alert event when all of the criteria occur in the defined time interval.
Top 3 Prices	Creates a window showing the top three distinct trade prices for each symbol. The example uses a Flex operator to create a custom operator with an embedded CCLScript script. A Flex operator creates a single output stream or window, and allows greater flexibility and control

	than a simple SELECT statement. The example also uses a named schema, which can be defined once and shared by the input stream and output window.
VWAP	Defines an input stream of stock market prices, as they might be reported from an exchange, and computes a moving average price called the volume weighted average price (VWAP). Uses a filter, and a simple aggregation (GROUP BY).

For details of each example, see the Samples subpage on the Welcome page.

See the *SAP HANA Smart Data Streaming: CCL Reference* for more code samples that demonstrate CCL and CCLScript.

In this section:

Related Information

About the Portfolio Valuation Sample Project [page 300]
SAP HANA Smart Data Streaming: CCL Reference

4.1.8.1 Loading a Sample Project

Load and run one of the example projects installed with smart data streaming to view end-to-end project execution in your workspace.

Prerequisites

- To run these examples, you may need to disable McAfee host intrusion prevention. See your McAfee documentation for details.
- 1. Click the **Customize Page** icon in the top right corner of the Welcome page to open the **Customize** dialog.
  2. Select the **Home** tab, and, in the **Root Pages** section, select the **Samples** option and click **OK**.

  Now you can access the example projects from the **Samples** subpage on the Welcome page.

Procedure

1. Navigate to the studio Welcome page.

2. Select the **Samples** subpage at the top of the page. Look at the samples and read their descriptions.
3. Click the name of the sample to load into your workspace. The SAP HANA Streaming Development perspective opens and loads the example.
   Project explorer shows all example projects, plus any other projects you have created.
4. Click in the main toolbar to run the project.
5. The sample opens in the SAP HANA Streaming Run-Test perspective. To load and run sample data through the project:
   a. Open the playback tab, located below the server view.
   b. Select the input data file.
   c. Click in the playback tab to run the sample data.

   Look at the various elements in the perspective to learn about the process.
   - Server view shows host and connection information, as well as the sample project, which is now subscribed to the example stream.
   - The console shows a series of status messages detailing server activity.
6. Open windows and streams to view published data in stream view.
   a. In server view, double-click each output window or output stream .
      Each stream or window opens in a separate tab. For example, the IndexesCalculation example opens four tabs, with one input stream and three output windows.
   b. Double-click and open input windows and input streams to view input data.

Next Steps

Run a second sample project. Server view now shows both samples. Expand it to show all streams for each project.

4.2 Cluster Connectivity

Before you can run a project in SAP HANA studio, first ensure that you are connected to an SAP HANA smart data streaming server.

A cluster consists of one or more workspaces, each with one or more projects. These projects can be running or stopped. All workspaces are within one server, which allows you to simultaneously work with multiple projects. You can also share a project within the cluster with other users. You can adjust the settings of the clustered projects through studio. However, you cannot use studio to change streaming node (host) settings.

When you click the run button, by default, studio will perform your most recent run operation. Alternatively, you can right-click to specify a different option.

In this section:

Modify the default preferences for how the machine running SAP HANA studio connects with other smart data streaming machines.

Configuring a Cluster Connection [page 65]
Set preferences to manage and run cluster connections with the smart data streaming plugin for SAP HANA studio.

Connecting to a Cluster [page 66]
Connect to an SAP HANA smart data streaming server from SAP HANA studio to be able to run a project.

Modifying a Cluster Connection [page 67]
Change the authentication settings of an existing, configured cluster connection.

Managing Secure Storage Credentials for a Streaming Server Connection [page 68]
Use secure storage to encrypt username and password credentials for an SAP HANA smart data streaming server connection in studio. Secure storage enables credential information for the streaming server connections to persist in studio.

4.2.1 Changing Networking Preferences

Modify the default preferences for how the machine running SAP HANA studio connects with other smart data streaming machines.

Context

SAP HANA studio sets the **Active Provider** to **Direct** to guarantee that network connections do not use a proxy server. If your network requires a different setting (such as the Eclipse default of setting it to **Native** if present, or **Manual** otherwise), modify the network preferences for studio.

> **Note**
>
> If the network is configured to use proxy servers, set the `no_proxy` environment variable on all publisher and subscriber machines in order to ensure successful publishing and subscribing of data.

Procedure

1. Open the studio.
2. Select **Preferences** > **General** > **Network Connections**.
3. Set the connection options as required for your network. If unsure, confirm the settings with your system or network administrator. To connect to a Web service through a proxy server:
   - Select **Native** from the *Active Provider* dropdown menu to use the system configured proxy settings.
   - Select **Manual** to configure your own proxy settings:

PUBLIC

- Set the proxy information for the HTTP or HTTPS schema.
- Set up proxy bypass for certain hosts. Add both the fully qualified domain name and short name of each host to the *Proxy bypass* list.

If you set *Active Provider* to **Native** or **Manual**, and you are connecting to the server using a native connection (directly rather than through a Web service), ensure both the fully qualified domain name and the short name of the server host are added to the *Proxy bypass* list. Otherwise, the connection may fail.

4. Click **Apply** to save your new settings, then click **OK**.

4.2.2 Configuring a Cluster Connection

Set preferences to manage and run cluster connections with the smart data streaming plugin for SAP HANA studio.

Prerequisites

The administrator of the cluster has provided the cluster host name and port number.

Procedure

1. To add a new cluster connection, open the SAP HANA Streaming Run-Test perspective, and select **New Server URL** in the server view toolbar.

   > **Note**
   >
   > Alternatively, in the server view toolbar, you can also select **Studio Preferences** and add a new connection through **SAP HANA smart data streaming** > **Run Test**, and click **Run Test**. Select **New**.

2. In the **New Server** dialog, enter the host name and port number for the cluster connection.

   Specify the port number using the format 3xx26, where xx is the instance number of your SAP HANA system. For example, if the instance number of your SAP HANA system is 00, enter the port number for the streaming server connection as 30026. Similarly, if your instance number is 14, then the port number becomes 31426.

3. Select a connection protocol: **Native**, or **Web Services**.
   - If you select **Web Services**, edit your network connection preferences. See Changing Networking Preferences [page 64] to configure a proxy server.
   - You can only connect through **Web Services** using the REST port. Before selecting this option, configure the Web Services Provider in the `wsp.xml` file. See *Configuring the Web Services Provider* in the *SAP HANA Smart Data Streaming: Adapters Guide*.
   - Connecting through a Web service is primarily intended for scenarios where a client is behind a firewall. Otherwise, use a native connection when possible, for performance reasons.

4. To enable encryption for Cluster Manager connections, select **SSL**.
5. Select User/Password as the authentication method. Kerberos and RSA are not supported for streaming clusters.
6. Click **OK**.

Results

In the SAP HANA Streaming Run-Test perspective, the server view accesses the list of stored server connections, and studio shows a login dialog for your clusters.

> **Note**
>
> To connect all listed servers, select **Reconnect All** in the server View toolbar.

Related Information

SAP HANA Smart Data Streaming: Adapters Guide
SAP HANA Smart Data Streaming: Configuration and Administration Guide

4.2.3 Connecting to a Cluster

Connect to an SAP HANA smart data streaming server from SAP HANA studio to be able to run a project.

Prerequisites

Start your SAP HANA system, and ensure the SAP HANA smart data streaming server on that system is running.

Procedure

1. Select the **SAP HANA Streaming Run-Test** perspective.
   The server view opens, displaying a list of the available clusters.
2. Right-click on the entry for the cluster you want (for example, myserver.mycompany.com:12345).
   Studio displays a pop-up menu.
3. Select **Connect Server**. If prompted, supply your SAP HANA username and password.
   The server view displays the workspaces on the cluster and the projects in each workspace.

4. Right-click on the project you want to run.
   Studio displays a pop-up menu.
5. Select **Show in** from the menu.
   Studio displays a pop-up menu listing ways to view the project's progress.
6. Select the viewing method, for example **Event Tracer View**.
   Studio starts displaying the project's progress in the specified view.

4.2.4 Modifying a Cluster Connection

Change the authentication settings of an existing, configured cluster connection.

Context

If the administrator of the cluster changes its authentication settings, modify the cluster connection in studio accordingly.

Procedure

1. In the server view of the SAP HANA Streaming Run-Test perspective, select **Studio Preferences** > **SAP HANA smart data streaming Studio** > **Run Test**.
   Studio displays the **Run Test** screen.
2. Select an existing server connection.
   The **Remove** and **Edit** buttons are activated.
3. Click **Edit**.
   Studio displays the **Remote Server Connection** screen.
4. Make your changes and click **OK**.
   Studio displays the **Run Test** screen.
5. Click **OK** to save your changes.

4.2.5 Managing Secure Storage Credentials for a Streaming Server Connection

Use secure storage to encrypt username and password credentials for an SAP HANA smart data streaming server connection in studio. Secure storage enables credential information for the streaming server connections to persist in studio.

Context

Store username and password credentials in secure storage for smart data streaming server connections. Secure storage stores credentials so that you do not have to enter username and password information each time you connect to a server. Studio stores the connections securely so your credentials are safe.

Enter username and password credentials in the following situations:

- The first time you connect to a server in studio.
- If you choose not to persist credential information.
- If the credential information in secure storage has been reset or removed.

When you connect to a server in studio, the **Set Credentials** prompt dialog contains the checkbox **Store Smart Data Streaming Credentials in Secure Storage**. Click the checkbox to store the username and password information you entered. If you do not choose to store the credentials using the **Store Smart Data Streaming Credentials in Secure Storage** option, you can store the credentials by opening the **Preferences** dialog.

Store credential information in secure storage.

Procedure

1. In the main menu, select **Windows** > **Preferences**.
2. Select **SAP HANA smart data streaming** > **Run-Test**.
3. Click on the checkbox for **Use Secure Storage for Streaming Credentials**.
4. Click **Apply** then **OK**.
   The credentials you entered when prompted are saved in `C:\Users\<username>\.eclipse\org.eclipse.equinox.security\secure_storage`.

Modifying Credentials in Secure Storage for a Streaming Server Connection

Context

Modify username and password credentials in secure storage for an SAP HANA smart data streaming server connection. Change username and password information as required.

Procedure

1. In the main menu, select **Windows** > **Preferences**.
2. Open the **Secure Storage** preference page. Either:
   - Select **Smart Data Streaming** > **Run-Test** and click on the **Secore Storage Settings** link, or,
   - Select **General** > **Secutiry** > **Secure Storage**.
3. Click on **UI Prompt**; then click on **Change Password**.
4. Modify the password information.

   Passwords are case-sensitive. You can use any alpha-numeric combination as well as symbols.
5. Click **Yes** or **No** for **Password Recovery**.

   Password Recovery prompts you to provide two questions and their expected answers. Once enabled, you can click on **Password Recovery**, answer the two questions, and recover your password, which avoids having to change the password. You can only access **Password Recovery** from the **Secure Storage** preference page.
6. Click **Finish**.

4.3 Developing Projects Using the SAP HANA Streaming Perspectives

Use either the visual editor or the CCL editor within the SAP HANA Streaming Development perspective to create and modify your projects. Start by developing a simple project, then test it iteratively as you gradually add greater complexity.

Context

You can create and edit a streaming project in the SAP HANA studio using either the visual editor or the CCL editor. When you create a new project, smart data streaming creates a `.ccl` file and a `.cclnotation` file automatically. You can edit either file in its respective editor; all changes are reflected in the parallel file. For example, if you add and configure an input adapter in the visual editor, then switch to the CCL editor, you see the ATTACH INPUT ADAPTER statement with all configured properties.

Procedure

1. Create a new streaming project. See Creating or Modify a Streaming Project in SAP HANA Studio [page 71].
2. Add input streams and windows. See Adding an Input Stream or Window to a Project [page 74].
3. Add output streams and windows with simple continuous queries. Gradually add complexity. See Simple Queries [page 177].
4. Attach adapters to streams and windows to subscribe to external sources or publish output. See Adding an Adapter to a Project [page 75].
5. (Optional) Use functions in continuous queries to perform mathematical operations, aggregations, datatype conversions, and other common tasks:
    - Built-in functions for many common operations
    - User-defined functions written in the CCLScript programming language
6. (Optional) Create named schemas to define a reusable data structure for streams or windows. See Working with Schemas [page 76].
7. (Optional) Create memory stores or log stores to retain the state of data windows in memory or on disk. See Data Retention and Recovery with Stores [page 158].
8. (Optional) Create modules to contain reusable CCL that can be loaded multiple times in a project. See Modularity [page 146]
9. Compile the project. See Compiling a Streaming Project [page 106].
10. Run the compiled project against test data, using the debugging tools in the smart data streaming in SAP HANA studio and command line utilities. See Running and Testing Projects using the SAP HANA Streaming Perspectives [page 108].

    Repeat this step as often as needed.

In this section:

Creating or Modify a Streaming Project in SAP HANA Studio [page 71]
Create, open, import, rename or delete streaming projects in the SAP HANA studio workspace directory.

Adding an Input Stream or Window to a Project [page 74]
You can add input streams and windows to a project, which accept data from a source external to the streaming project.

Adding an Adapter to a Project [page 75]
Attach an adapter to a project by inserting it into the project diagram, connecting it to a stream or window, and setting properties.

Working with Schemas [page 76]
Discover external schemas and create CCL schemas, streams, or windows based on the format of the data from the data source that is connected to an adapter.

Specifying a Retention Policy [page 87]
The keep policy determines the basis for retaining rows in a window.

4.3.1 Creating or Modify a Streaming Project in SAP HANA Studio

Create, open, import, rename or delete streaming projects in the SAP HANA studio workspace directory.

Procedure

1. In the SAP HANA studio menu bar, go to **File** > **New** > **Project...**, then select **Smart Data Streaming** > **New Streaming Project**.
2. In the **Name** field, enter a valid project name. Create a name that:
   - Starts with a lowercase letter, underscore, or dollar sign.
   - Uses lowercase letters, numbers, underscores, or dollar signs for all other characters.
   - Cannot contain spaces.
3. In the **Directory** field, accept the default location or browse to a directory in which to store the new project folder.
4. Click **Finish** to create the project files.

The new project opens in the visual editor with one input stream, NEWSTREAM, and an inline schema ready for editing.

Opening a Streaming Project in SAP HANA Studio

Procedure

1. In the SAP HANA studio menu bar, click **Windows** > **Open Perspective** > **Other** > **SAP HANA Streaming Development** and click **OK**.
2. In the studio toolbar, click the **Open Streaming Project** icon.
3. Browse to the root directory of the project.
4. (Optional) Select **Copy projects into workspace** to copy and open the project in the workspace. Changes are made to the copy only.
   If you do not select this option, the project opens in its original location.
5. Click **Finish**.

Importing a Streaming Project into SAP HANA Studio

Procedure

1. In the SAP HANA studio menu bar, click **Windows** > **Open Perspective** > **Other**.
2. From the **Open Perspective** dialog, select **SAP HANA Streaming Development** and click **OK**.
3. In the **SAP HANA Streaming Development** perspective, right-click the project explorer view, and select **Import** from the context menu.
4. In the Import dialog, expand the **General** folder and click **Existing Projects into Workspace**, then click **Next**.
5. Enable the **Select root directory option** and enter or browse to the root directory containing the projects you are importing.
6. (Optional) Clear the check mark from any projects you do not want to import.
7. (Optional) Clear the **Copy projects into workspace** option.
8. Click **Finish**.

Renaming a Project

Procedure

1. Select the project folder in the project explorer.
2. Right click and select **Rename**.
   The same naming rules that apply when creating a project also apply when renaming a project:
   - Starts with a lowercase letter, underscore, or dollar sign.
   - Uses lowercase letters, numbers, underscores, or dollar signs for all other characters.
   - Cannot contain spaces.
3. In the **Rename Resource** field, enter the new project name.
   All three of the files in the project folder are automatically renamed:

<project_name>.ccl	contains the CCL code.
<project_name>.cclnotation	contains the diagram that corresponds to the `.ccl` file.
<project_name>.ccr	contains the project configuration.

4. Click **OK**.
   You will see the name change reflected in the project folder and the three project files.

Deleting a Project

Procedure

1. In the project explorer and select a project, or multiple projects.
2. Right click and select **Delete**.
3. (Optional) In the dialog, click **Preview** to view a list of the selected projects. If you selected multiple projects to delete, the preview option allows you to uncheck any projects you want to keep.
4. (Optional) Check the **Delete project contents on disk** option to completely erase projects from the smart data streaming workspace and from your machine.
   If this option is not checked, smart data streaming deletes the project from the project explorer, but does not remove the project from your workspace directory.
5. Click **OK**.

4.3.2 Adding an Input Stream or Window to a Project

You can add input streams and windows to a project, which accept data from a source external to the streaming project.

Context

You can create an input stream or window by adding an adapter that supports schema discovery, and generating the stream or window to inherit the schema of the external data source automatically. You can then add columns, and specify whether they need an autogenerate clause, which automatically generates data for specified columns.

Procedure

1. From the SAP HANA Streaming Development perspective, in the palette menu of the visual editor, open the **Streams and Windows** category and select either:
    - **Input Stream**
    - **Input Window**
2. Select a location in the diagram and click to add the shape.
3. To set the name of the input stream or window, either:
    - Click to edit the shape name, or,
    - In verbose mode, click the **Edit** icon next to the name.

    > **Note**
    >
    > When you create a duplicate named window or stream in the visual editor, then save your file and switch to the CCL editor, a third copy of the original stream or window is created. You can see this third copy only when you switch back to the visual editor. To remove it, click **Remove all shapes from diagram** to clear out all the shapes, then click **Add all shapes** to bring back the original stream or window, and the duplicate stream or window.
4. Click **Add Column** to add each new column to the schema, then set key columns and edit column expressions.
5. (Optional) To delete columns, select them and press **Delete**.
6. (Optional for windows, not permitted for streams) Select **Set Keep Policy** and choose an option.
7. (Optional) Double-click the policy to edit its parameters.
8. (Optional for both windows and streams) Select **Set Autogenerate**, choose the columns from the Candidate list (only columns with a `long` datatype will populate the Candidate list) and click **Add**.

> **Note**
>
> You can also manually specify a column to add to the autogenerate list by clicking **Add Column** and entering in a column name. You can only use columns with the `long` datatype.

9. (Optional) To remove columns from the autogenerate list, select them and click **Remove**.
10. To set a From value for the autogenerate clause to start with, click **Select** and choose a variable or parameter from the list. You can also manually enter a variable or parameter that is used within a declare block of a column with a `long` datatype.
11. Click **OK**.

4.3.3 Adding an Adapter to a Project

Attach an adapter to a project by inserting it into the project diagram, connecting it to a stream or window, and setting properties.

Procedure

1. From the SAP HANA Streaming Development perspective, open the **Input Adapters** or **Output Adapters** compartment in the palette.
2. Click an adapter shape in the palette, then click in the diagram.
3. Attach the adapter to a stream or window. Either:
    - Generate and attach the stream or window automatically, using schema discovery (best practice for adapters that support it), or,
    - Create the stream or window, then attach it to an:
        - **Input adapter:** click the **Connector** tool, then click the adapter shape in the diagram, then click the stream or window.
        - **Output adapter:** click the **Connector** tool, then click the stream or window in the diagram, then click the adapter shape.
4. (Optional) Edit the adapter name.
5. (Optional) Edit the adapter properties. Choose one of the following:
    - Select **Consolidate Adapter Properties from the adapter properties dialog**. When this setting is enabled, the system looks for matching property pairings in the CCR, and removes them from the adapter properties collection. This setting is only relevant when an adapter has a property set configured; however, it is a global setting that affects all such adapters, not just the one you are currently attaching to the project.
    - Select **Use named property set** to use a named property set from the project configuration file, then configure any properties that are not included in the property set.
    - In the table, modify adapter properties manually.

Related Information

Adapters [page 30]

4.3.4 Working with Schemas

Discover external schemas and create CCL schemas, streams, or windows based on the format of the data from the data source that is connected to an adapter.

Every row in a stream or window has the same structure (schema), including the column names and datatypes and the order in which the columns appear. Multiple streams or windows can use the same schema, but each stream or window can only have one schema.

Rather than manually creating a new schema in your smart data streaming project, you can use schema discovery to discover and automatically create a schema, stream, or window based on the format of the data from the data source to which your adapter connects. For example, if you create a table in your SAP HANA database, use the SAP HANA Output adapter to connect to the database. You can then use schema discovery to discover and create a schema, stream, or window in your smart data streaming project that corresponds to the schema of the table you created in your SAP HANA database.

While using discovery is a convenient way to create your CCL schema, pay attention to the datatypes that the CCL columns inherit from the external data source. Discovery tries to maintain the same level of precision, or greater, when mapping source datatypes to smart data streaming datatypes. Some databases, such as SAP IQ, support microsecond precision for the `SQL_TIMESTAMP` and `SQL_TYPE_TIMESTAMP` datatypes. As such, schema discovery maps these types to the smart data streaming datatype `bigdatetime`, which also supports microsecond precision. If your smart data streaming project does not require this level of precision, you can, after generating your schema through discovery, modify the schema to use a lower-precision datatype, such as `msdate` for millisecond precision.

To enable schema discovery, configure the properties of the adapters that support the feature.

In this section:

Discovering a Schema [page 77]
Use the **Schema Discovery** button in the visual editor to discover and automatically create a schema based on the format of the data from the adapter.

Discovering Schema and Creating a Mapping File for the SAP RFC Adapter [page 79]
Use the **Schema Discovery** button in the visual editor to discover function, table, or query schema and create a mapping file for the RFC adapter.

Discovering Schema and Creating a Mapping File for the Web Services (SOAP) Adapter [page 80]
Use the **Schema Discovery** button in the visual editor to discover schema and create a mapping file for the Web Services (SOAP) adapter.

Creating a Schema in the Visual Editor [page 81]
Create a shared schema object that can be referenced from any number of streams or windows.

Creating a Schema in the CCL Editor [page 82]
Enter a CREATE SCHEMA statement using the CCL editor to provide users with a shared schema object that can be referenced from any number of streams or windows.

Adapter Support for Schema Discovery [page 82]
Some of the adapters that come with smart data streaming support schema discovery, while others do not. If you use an adapter that supports schema discovery, you need to set certain properties.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide

4.3.4.1 Discovering a Schema

Use the **Schema Discovery** button in the visual editor to discover and automatically create a schema based on the format of the data from the adapter.

Prerequisites

- For a database adapter, you need access to the database on the system from which you are using SAP HANA studio.
- Define a data service using the smart data streaming data services view, and add the adapter to the diagram.
- When using the SAP HANA smart data streaming plugin for SAP HANA studio, there is no default cluster. Define a cluster in the server view of the SAP HANA Streaming Run-Test perspective before you can update the Default Server URL preference, or the server will not be available to select. See Connecting to a Cluster [page 66].
- Discovery uses the Default Server URL preference to retrieve the list of available data services. The default server URL is esp://localhost:9786. Because there is no localhost smart data streaming server, to locate available data services, update the Default Server URL preference:
    1. In the SAP HANA Streaming Run-Test perspective, select **Window** > **Preferences**.
    2. In the Preferences dialog, select **Smart Data Streaming**.
    3. In the **Default Server URL** field, click **Change** and select a server from the Select Default Server URL dialog. Click **OK**.
    4. In the Preferences dialog, click **Apply**, then **OK**.

Procedure

1. In the SAP HANA Streaming Development perspective, open or create a project with an input adapter and configure the adapter for schema discovery. In the adapter shape, click **Edit Properties** and complete the dialog:
    - Select **Use named property set** and select a property set from the drop down menu, or,

- Select **Consolidate adapter properties from 'Adapter Properties' dialog** and enter property values in the Basic and (optionally) Advanced tabs. Required properties are in red.

For example, to use schema discovery for the File CSV Input adapter, first configure the directory and file properties for the adapter, to specify the absolute path to the data files you want the adapter to read.

> **Note**
>
> To create a named property set, edit adapter properties in the project configuration file.

2. Click **Schema Discovery** on the adapter toolbar.
    - If the schema is successfully discovered, a dialog appears where you can view and select a schema.
    - If the schema is not successfully discovered, an error message appears stating that no schema was discovered for the adapter. You can:
        - Check that the adapter properties are configured for schema discovery.
        - Check to see if the adapter supports schema discovery.
3. Select a schema, and click **Next**.
4. In the dialog for creating an element, select an option:

Adapter State	Available Options
The adapter is not attached to a stream or window.	○ **Create a new input stream (with inline schema)**: creates and attaches a new stream to the adapter, creates an inline schema for the stream, and populates the stream with the schema discovered from the adapter. ○ **Create a new input window (with inline schema)**: creates and attaches a new window to the adapter, creates an inline schema for the window, and populates the window with the schema discovered from the adapter. ○ **Create a new input stream (with attached schema)**: creates and attaches a new stream to the adapter, creates and attaches a new named schema to the stream, and populates the stream with the schema discovered from the adapter. ○ **Create a new input window (with attached schema)**: creates and attaches a new window to the adapter, creates and attaches a new named schema to the window, and populates the window with the schema discovered from the adapter. ○ **Create a new named schema**: creates a new named schema and populates it with the schema discovered from the adapter.
The adapter is already attached to a stream or window.	○ **Apply the schema to the connecting stream or window**: populates the stream or window with the schema discovered from the adapter. ○ **Create a new named schema:** creates a new named schema and populates it with the schema discovered from the adapter.

5. Click **Finish**.
    - The mapping file you specified in the Adapter Mapping File property is populated with mappings based on the schema you selected.
    - Either the window or stream that is attached to the adapter is populated with the schema you selected or a new named schema is created in the project to which the adapter is attached.
6. (Optional) Create or modify a mapping of adapter columns to smart data streaming columns.
    a. In the adapter shape, click **Edit Properties** .
    b. In the Adapter Properties screen, click the **Advanced** tab.
    c. Click the **Value** column of the **Field Mapping** row.
    The system displays an ellipsis in that field.
    d. Click on the ellipsis.
    The system displays the **Define Adapter Field Mapping (Permutation)** screen.

e. Click the **Column** field next to the database column you want to map to your smart data streaming column.
A down arrow is displayed, indicating a dropdown list of choices is available.
f. Click the mouse in the entry field to display the dropdown list and select the smart data streaming column to which you wish to map the database column.
g. Clear the check boxes next to any database columns that you do not wish to map (only checked columns will be mapped) and click **OK**.
Studio removes the dialog and redisplays the Adapter Properties screen with the new mapping in the **Field Mapping Value** column.
h. You can also click **Select All** to place a check in all the database column check boxes or **Remove All** to remove the check from all of the check boxes.

4.3.4.2 Discovering Schema and Creating a Mapping File for the SAP RFC Adapter

Use the **Schema Discovery** button in the visual editor to discover function, table, or query schema and create a mapping file for the RFC adapter.

Prerequisites

Add the SAP RFC adapter to the diagram.

Procedure

1. In the SAP HANA Streaming Development perspective, configure the RFC Input or Output adapter for schema discovery. In the adapter shape, click **Edit Properties** and set these properties:
   - Adapter Configuration File
   - Adapter Mapping File
   - SAP Host
   - SAP System Number
   - SAP Client
   - Username
   - Password

   Ensure there are no checkboxes selected, and click **OK**.
2. Click **Schema Discovery** on the adapter toolbar.
   - If the schema is successfully discovered, a dialog appears where you can view and select schemas.
   - If the schema is not successfully discovered, an error message appears stating that no schema was discovered for the adapter. Check that the adapter properties are configured for schema discovery and that no checkboxes are selected in the Edit adapter properties dialog.

3. In the schema discovery dialog:
    - (RFC Input adapter only) Select which remote schemas to search: **Functions**, **Tables**, or **Queries**. Scroll through to view the discovered functions, tables, or queries.
    - (RFC Output adapter only) Scroll through to view the discovered schemas. Only function schemas display.
4. Select a function, table, or query and click **Select Schema**.
5. Click **Next**.
6. In the Create Element dialog, select an option:
    - **Assign schema to the connecting stream/window**: populates the stream or window to which the adapter is attached with the selected adapter schema.
    - **Create new named schema**: creates a new named schema in smart data streaming which is made up of the RFC schema elements.
7. Click **Next**.
8. From the left column, select the remote fields for your schema and click **Select Field(s)**.
These fields now appear under the Selected Mapping fields column. To remove any fields from this column, select the field and click **Select Field(s)** again.
9. Click **Finish**.
    - The mapping file you specified in the Adapter Mapping File property is populated with mappings based on the schema you selected.
    - Either the window or stream that is attached to the adapter is populated with the schema you selected or a new named schema is created in the project to which the adapter is attached.

Next Steps

- For BW mode, edit the generated mapping file by adding the `<variables>` element.
- For Generic RFC mode, edit the generated mapping file by adding the `<variables>` and `<input>` elements.

4.3.4.3 Discovering Schema and Creating a Mapping File for the Web Services (SOAP) Adapter

Use the **Schema Discovery** button in the visual editor to discover schema and create a mapping file for the Web Services (SOAP) adapter.

Prerequisites

Add the Web Services (SOAP) adapter to the diagram.

Procedure

1. In the SAP HANA Streaming Development perspective, configure the Web Services (SOAP) Input or Output adapter for schema discovery. In the adapter shape, click **Edit Properties** and set these properties:
   - Adapter Configuration File
   - Adapter Mapping File
   - Discovery WSDL URL
   - Discovery Working Directory
   - Discovery Service Name

   Ensure there are no checkboxes selected, and click **OK**.
2. Click **Schema Discovery** on the adapter toolbar.
   - If the schema is successfully discovered, a dialog appears where you can view and select schemas.
   - If the schema is not successfully discovered, an error message appears stating that no schema was discovered for the adapter. Check that the adapter properties are configured for schema discovery and that no checkboxes are selected in the Edit Adapter Properties dialog.
3. Select a discovered schema and click **Next**.
4. In the Create Element dialog, select an option:
   - **Assign schema to the connecting stream/window**: this populates the stream or window to which the adapter is attached with the selected adapter schema.
   - **Create new named schema**: this creates a new named schema in smart data streaming which is made up of the Web Services (SOAP) schema elements.
5. Click **Finish**.
   - The mapping file you specified in the Adapter Mapping File property is populated with mappings based on the schema you selected.
   - Either the window or stream that is attached to the adapter is populated with the schema you selected or a new named schema is created in the project to which the adapter is attached.

4.3.4.4 Creating a Schema in the Visual Editor

Create a shared schema object that can be referenced from any number of streams or windows.

Procedure

1. From the SAP HANA Streaming Development perspective, in the palette menu under the Shared Components category, select **Named Schema** ().
2. Click anywhere in the visual editor to place the schema.
3. Set the name of the schema by either:
   - Double-clicking the name label, or,

- Editing the name field from within the Properties window.

4. Click **Add Columns** ( ) to add individual columns.
5. Edit column names and datatypes.

> **Tip**
>
> When entering column names and their datatypes, use **Tab** to easily move between cells in the table.

6. (Optional) Connect the schema to one or more streams or windows using the connector tool.

4.3.4.5 Creating a Schema in the CCL Editor

Enter a CREATE SCHEMA statement using the CCL editor to provide users with a shared schema object that can be referenced from any number of streams or windows.

Procedure

In the CCL editor, enter valid CCL for the CREATE SCHEMA statement.

- Enter text manually.
- Choose **Create** > **Schema**, and edit the draft CCL code as needed.

For example, this statement creates a shared schema object named SchemaTrades1, with four columns:

```
CREATE SCHEMA SchemaTrades1 (
Symbol STRING ,
Seller STRING ,
Buyer STRING ,
Price FLOAT )
```

4.3.4.6 Adapter Support for Schema Discovery

Some of the adapters that come with smart data streaming support schema discovery, while others do not. If you use an adapter that supports schema discovery, you need to set certain properties.

For additional details on the adapter properties, see the specific adapter section.

Adapter	Supports Schema Discovery	Required Properties
Database Input	Yes	Database Service Name of database service from which the adapter obtains the database connection.

Adapter	Supports Schema Discovery	Required Properties
Database Output	Yes	Database Service Name of service entry to use.
File/Hadoop CSV Input	Yes	File Input Transporter: • Dir • File • AccessMode • (Optional) ScanDepth CSV String to Streaming Formatter • ExpectStreamNameOpcode
File/Hadoop CSV Output	No	—
File/Hadoop JSON Input	No	—
File/Hadoop JSON Output	No	—
File/Hadoop XML Input	No	—
File/Hadoop XML Output	No	—
File/Hadoop XML Event Input	Yes	File Input Transporter: • Dir • File • AccessMode • (Optional) ScanDepth
File/Hadoop XML Event Output	No	—
File FIX Input	No	—
File FIX Output	No	—
FTP CSV Input	No	—
FTP CSV Output	No	—
FTP Event XML Input	No	—
FTP Event XML Output	No	—
HTTP Client Output	No	—
HTTP Output	No	—

Adapter	Supports Schema Discovery	Required Properties
JDBC Input	Yes	JDBC Input Transporter: • Host • Port • User • Password • DbName • DbType • DbDriver
JDBC Output	Yes	JDBC Output Transporter: • Host • Port • User • Password • DbName • DbType • DbDriver
JMS CSV Input	Yes	JMS Input Transporter: • ConnectionFactory • JndiContextFactory • JndiURL • DestinationType • DestinationName • MessageType • (Optional) ScanDepth
JMS CSV Output	No	—
JMS Event XML Input	Yes	JMS Input Transporter: • ConnectionFactory • JndiContextFactory • JndiURL • DestinationType • DestinationName • MessageType • (Optional) ScanDepth
JMS Event XML Output	No	—
JMS FIX Input	No	—
JMS FIX Output	No	—
JMS MAP Input	No	—
JMS MAP Output	No	—

Adapter	Supports Schema Discovery	Required Properties
JMS Object Input	No	—
JMS Object Output	No	—
Log File Input	No	—
Kafka Avro Record Input	No	—
Kafka Avro Record Output	No	—
Kafka CSV Input	Yes	Kafka Input Transporter: • `kafkaBootstrapServers` or `kafkaConsumerProperties` with the `bootstrap.servers` property set in the consumer property file
Kafka CSV Output	No	—
Kafka Event XML Input	Yes	Kafka Input Transporter: • `kafkaBootstrapServers` or `kafkaConsumerProperties` with the `bootstrap.servers` property set in the consumer property file
Kafka Event XML Output	No	—
Kafka JSON Input	No	—
Kafka JSON Output	No	—
Kafka String Input	No	—
Kafka String Output	No	—
Random Tuples Generator Input	No	—
Replication Server Input	Yes	• RSSD Host • RSSD Port • RSSD Database Name • RSSD User Name • RSSD Password
SAP ASE Output	Yes	DB Service Name The name of the database service that represents the SAP ASE database into which information will be loaded.
SAP HANA Output	Yes	DB Service Name The name of the database service that represents the SAP HANA database into which information will be loaded.

Adapter	Supports Schema Discovery	Required Properties
SAP IQ Output	Yes	DB Service Name The name of the database service that represents the IQ database into which information will be loaded.
SAP RFC Input	Yes	• Adapter Configuration File • Adapter Mapping File • SAP Host • SAP System Number • SAP Client • Username • Password
SAP RFC Output	Yes	• Adapter Configuration File • Adapter Mapping File • SAP Host • SAP System Number • SAP Client • Username • Password
SMTP Output	No	—
Socket CSV Input	No	—
Socket CSV Output	No	—
Socket Event XML Input	No	—
Socket Event XML Output	No	—
Socket FIX Input	No	—
Socket FIX Output	No	—
Socket JSON Input	No	—
Socket JSON Output	No	—
Tibco Rendezvous Input	No	—
Tibco Rendezvous Output	No	—
Web Services (SOAP) Input	Yes	• Adapter Configuration File • Adapter Mapping File • Discovery WSDL URL • Discovery Working Directory • Discovery Service Name

Adapter	Supports Schema Discovery	Required Properties
Web Services (SOAP) Output	Yes	• Adapter Configuration File • Adapter Mapping File • Discovery WSDL URL • Discovery Working Directory • Discovery Service Name
WebSphere MQ Input	No	—
WebSphere MQ Output	No	—
Streaming Web Services Provider	No	—
Streaming Web Output	No	—

4.3.5 Specifying a Retention Policy

The keep policy determines the basis for retaining rows in a window.

You can set a keep (or retention) policy for any window with a memory-based store, including any simple query that produces a window.

Retention policies for windows that use a log store are only supported for input windows.

The following table describes the options for keep policies:

Options	Description
All rows	Retain all rows in the window (default).
Last row	Retain only the last row in the window.

Count	Either:
	• Enter the absolute number of rows to retain, or,
	• Choose **Select** and select a previously declared variable or parameter to determine a specific range of rows to retain in the window.
	
	Tip
	If the list is empty and you want to base the count on a parameter or variable, switch to the CCL editor and define it in a DECLARE block at the beginning of the CCL. For example:
	`DECLARE`
	`integer test :=50;`
	`end;`
	Then go back and select it.

Options	Description
Every	(Optional) Works with the Count and Time options. When used with the Count option, Every retains a number of rows based on the Count value specified, and purges all of the retained rows once a row arrives that would exceed the specified maximum number of rows. This purge occurs after the specified Count number has been reached. When used with the Time option, Every retains a number of rows within a specified time interval. Once the time interval expires, all rows are purged simultaneously. **Note** When this option is used, the resulting retention is based on a jumping window policy. Otherwise, the resulting retention is based on a sliding window policy.
Slack	For a count-based policy, set the number of rows to delete when the maximum number of rows is reached (the Count value). Default is 1, that is, when the window contains `<count-value>` rows, each new row causes the oldest row to be deleted. Setting Slack to greater than 1 can optimize performance.
Time	Set a time limit on the window, and specify a time period to determine what age of row to retain in the window. Press **Ctrl+Space** to choose the unit of time.
PER clause	(Optional) Works with the Time and Count options. When used with the Count option, PER works in conjunction with the specified Count number to retain the Count number of rows across each column specified under the PER clause. When used with the Time option, PER works in conjunction with the specified Time interval to retain the rows within that Time interval across each column specified under the PER clause. List the names of the columns that need to be retained in the PER clause box, with a comma separating each column name entered.

Count

In a sliding window count-based retention policy, a constant integer specifies the maximum number of rows retained in the window. To retain the specified maximum number of rows in the window, the policy purges the oldest entries as new entries arrive, one row at a time.

In a jumping window count-based retention policy, enabled by using the Every option, all rows are purged only once a row arrives that would exceed the specified maximum number of rows.

A sliding window count-based policy also defines an optional slack value, which can enhance performance by requiring less frequent cleaning of memory stores.

Slack

Slack is an advanced feature used to enhance performance by requiring less frequent cleaning of memory stores. It sets a maximum of N + S rows in the window, where N is the retention size (the count setting) and S is the slack. When the window reaches N + S rows, the systems purges S rows. The larger the value of slack the better the performance is, since there is less cleaning required.

The default value for Slack is 1. When slack = 1, after the window reaches the maximum number of records, each time a new record is inserted, the oldest record is deleted. This causes a significant impact on performance. When slack > 1, say Y, then the window will accumulate up to X + Y number of records. The next record inserted will then cause the deletion of Y records. Larger slack values improve performance by reducing the need to constantly delete rows.

> **Note**
>
> You cannot use slack with the Every option.

Time

In a sliding window time-based retention policy, a time interval specifies the maximum age of the rows retained in the window. Rows are purged from the window, one row at a time, when they become older than the specified interval.

In a jumping window time-based retention policy, enabled by using the Every option, all rows produced in the specified time interval are purged after the interval has expired.

PER Clause

The PER clause allows for rows specified by the Count or Time options to be retained across specified columns.

For a count-based retention policy, it keeps the number of rows specified by the Count number across each column specified under the PER clause. The rows in each column specified to be retained will update simultaneously to delete older entries as newer ones arrive.

For a time-based retention policy, it keeps rows within the specified Time interval across each column specified under the PER clause. The rows in each column specified to be retained will update simultaneously to delete older entries as the time interval expires.

Related Information

Streams and Windows [page 19]
Creating and Modifying Aggregate Queries [page 195]
Creating and Modifying Join Queries [page 186]

4.3.6 Setting Key Columns

Set primary keys in the visual editor within the column compartment of the window, and Flex operator shapes.

Context

Multiple columns can be designated as primary keys. In the visual editor, primary keys appear as icons. Deduced primary keys, which appear as icons, are calculated when the PRIMARY KEY DEDUCED flag is set for the target element.

> **Note**
>
> Only windows support PRIMARY KEY DEDUCED. You can modify the deduced key property for these elements from the Properties view.

The ability to set key columns and view key column icons as described here does not apply when using column expressions.

Procedure

1. From the SAP HANA Streaming Development perspective visual editor, expand the **Columns** compartment of the desired query object (window, or Flex shape).
2. Click the icon to the left of the column name to make it a primary key.

   A single-key icon now designates the column as a primary key.
3. (Optional) To set a primary key for query objects with a deduced primary key, click any column or deduced key within the target window.
   The column you initially selected and all other deduced key columns are now primary keys. In addition, the target window is no longer PRIMARY KEY DEDUCED.
4. (Optional) To remove the primary key designation from a column, click to the left of the column name.
   A column icon replaces the single key icon, indicating that the column is no longer part of the primary key.

4.3.7 Editing Column Expressions

Modify column expressions for windows and streams using an inline editor or dialog-based expression editor.

Procedure

1. (Optional) To add a column expression, click **Add Column Expressions** in the shape toolbar.
2. Expand the **Column Expressions** compartment.
3. To modify a column expression, choose one of the following options:
   - Double-click to open the inline editor. Type into the edit box to edit the existing expression or enter a new one. Press **Ctrl+Space** for a list of available columns and functions.
   - Press **Ctrl+F2** to open the expression editor. Press **Ctrl+Space** to show the available input columns and built-in functions, or manually enter the expression.
   - Modify the expression in the Properties view.

   Tip

   When entering column names and their datatypes, use **Tab** to easily move between cells in the table.

Related Information

CCL Functions [page 32]

4.3.8 Creating Windows and Streams from a Database Table

Create windows and streams from SAP HANA tables for a smart data streaming project in studio.

Procedure

1. In SAP HANA studio, open the SAP HANA Streaming Development perspective, and select the systems view or data services view.
2. Select an SAP HANA table.
3. Drag and drop the SAP HANA table into the project workspace.
4. From the **Create CCL from HANA Table** dialog, click the **Service** button, and select a data service.
5. From **Create Type**, specify the type of stream or window to create:
   - Input Stream

- Output Stream
- Input Window
- Output Window

6. If you selected output stream or output window in the previous step, specify the **Output Adapter Type** as:
   - Generic Database Output (default)
   - Hana Database Output
7. Select a **Schema**:
   - Named Schema Assignment
   - Inline
8. Click **OK**.

4.3.9 Using the SAP HANA Repository

There are various advantages to placing your smart data streaming projects into the SAP HANA repository, including version control, and the ability to share your project files with other team members and create delivery units. Create a new smart data streaming project and share it to create a link between your development workspace and the SAP HANA repository.

Procedure

1. Create a new smart data streaming project in the SAP HANA studio.
2. Share the project to create a link between project-specific files in your development workspace and the SAP HANA repository. See *Tutorial: Share an Application Project* in the *SAP HANA Developer Guide* .

   Saving the project in your workspace automatically saves it to the repository.
3. Compile your project when you are ready to test it.

   At this point, you can create a delivery unit. See *Create a Delivery Unit* in the *SAP HANA Developer Guide* .
4. Right-click on your project folder, select **Team** > **Activate** to activate, and automatically deploy your project to the smart data streaming server.

Results

Switch to the SAP HANA Streaming Run-Test perspective to verify that the project is running.

If the project is not running after one minute and does not appear in the SAP HANA Streaming Run-Test perspective, use the XS Job Dashboard in the SAP HANA XS Administration Tool to locate the `sap.hana.streaming.repoplugin::plugin.xsjob` job and troubleshoot errors. See *The XS Job Dashboard* in the *SAP HANA Administration Guide* for additional details. Make any necessary changes to your project CCL file, then recompile and reactivate the project.

Related Information

SAP HANA Developer Guide
SAP HANA Administration Guide

4.3.10 Project Configurations

A project configuration is an XML document that governs specific runtime properties of a project, including stream URI bindings, adapter properties, parameter values, and advanced deployment options.

Project configuration files are created and edited separately from the project they are attached to, and are identified by their `.ccr` file extension. View and edit project configuration files in the project explorer view in the SAP HANA Streaming Development perspective.

Configuration files maintain all runtime properties outside the CCL. Thus, you can maintain CCL and CCX files under version control, while varying runtime properties. This lets you move a project from a test environment to a production environment without modifying the CCL and CCX files.

By default, when you create a new project, smart data streaming also creates a new project configuration file. One project may have multiple configuration files attached to it, so you can manually create new project configurations. However, opening your project using the smart data streaming plugin for SAP HANA studio only deploys the original CCR file created with the project, and this CCR file has the same name as the CCX file. To deploy your project with a separate CCR file, launch the project using the `streamingclusteradmin` utility from the command line. For more information on how to add and run projects from the command line, see *streamingclusteradmin in Command Line Mode* in *SAP HANA Smart Data Streaming: Utilities Guide*.

In this section:

Sample Project Configuration File [page 105]
If you prefer using text editor instead of the Studio Project Configuration editor, use this example to help you build and modify your own project configuration (`.ccr`) files.

4.3.10.1 Creating a Project Configuration

When you create a new project, a project configuration file is automatically generated. However, you can create additional project configuration files and edit configuration properties.

Procedure

1. From the smart data streaming plugin for SAP HANA studio, select **File** > **New** > **Other...**, and, from the **New** dialog, expand the SAP HANA smart data streaming folder and select **CCR Project Configuration**.
2. Select the folder in which to store the new configuration file, and assign it a file name.
3. Click **Finish.**
   The CCR Project Configuration editor window appears.

4.3.10.2 Opening an Existing Project Configuration

Open an existing project configuration file.

Procedure

1. From the smart data streaming plugin for SAP HANA studio, select **Window** > **Open Perspective** > **Other...**, and, from the **Open Perspective** dialog, select **SAP HANA Streaming Development**.
2. Select **Window** > **Show View** > **Project Explorer**.
3. Locate the project configuration file, which appears as `<project-name>.ccr`, and double-click to open.

4.3.10.3 Datatypes

SAP HANA smart data streaming supports integer, float, string, money, long, and timestamp datatypes for all of its components.

Datatype	Description
`bigdatetime`	Timestamp with microsecond precision. The default format is YYYY-MM-DDTHH:MM:SS:SSSSSS. All numeric datatypes are implicitly cast to `bigdatetime`. The rules for conversion vary for some datatypes: • All `boolean`, `integer`, and `long` values are converted in their original format to `bigdatetime`. • Only the whole-number portions of `money(n)` and `float` values are converted to `bigdatetime`. Use the `cast` function to convert `money(n)` and `float` values to `bigdatetime` with precision. • All `seconddate` values are multiplied by 1000000 and converted to microseconds to satisfy `bigdatetime` format. • All `msdate` values are multiplied by 1000 and converted to microseconds to satisfy `bigdatetime` format.
`bigint`	An alias for `long`.
`binary`	Represents a raw binary buffer. Maximum length of value is platform-dependent, with a size limit of 2 gigabytes. NULL characters are permitted.
`boolean`	Value is true or false. The format for values outside of the allowed range for `boolean` is 0/1/false/true/y/n/on/off/yes/no, which is case-insensitive.
`seconddate`	Date with second precision. The default format is YYYY-MM-DDTHH:MM:SS.
`decimal`	Used to represent numbers that contain decimal points. Accepts `precision` and `scale`, two mandatory parameters that determine the range of values that can be stored in a `decimal` field. `precision` specifies the total number (from 1 to 34) of digits that can be stored. `scale` specifies the number of digits (from 0 to `precision`) that can be stored to the right of the decimal point. The value 88.999p10s3 would have a decimal datatype of (10,3), which means the value has a decimal precision of 10 and a decimal scale of 3.
`double`	A 64-bit numeric floating point with double precision. The range of allowed values is approximately -10^{308} through $+10^{308}$. Equivalent to `float`.
`float`	A 64-bit numeric floating point with double precision. The range of allowed values is approximately -10^{308} through $+10^{308}$. Equivalent to `double`.

Datatype	Description
`integer`	A signed 32-bit integer. The range of allowed values is -2147483648 to +2147483647 (-2^{31} to 2^{31}-1). Constant values that fall outside of this range are automatically processed as long datatypes. To initialize a variable, parameter, or column with a value of -2147483648, specify (-2147483647) -1 to avoid CCL compiler errors.
`interval`	A signed 64-bit integer that represents the number of microseconds between two timestamps. Specify an `interval` using multiple units in space-separated format, for example, "5 Days 3 hours 15 Minutes". External data sent to an interval column is assumed to be in microseconds. Unit specification is not supported for `interval` values converted to or from `string` data. When an `interval` is specified, the value must fit in a 64-bit integer (`long`) when it is converted to the appropriate number of microseconds. For each `interval` unit, the maximum allowed values that fit in a `long` when converted to microseconds are: • MICROSECONDS (MICROSECOND, MICROS): +/- 9223372036854775807 • MILLISECONDS (MILLISECOND, MILLIS): +/- 9223372036854775 • SECONDS (SECOND, SEC): +/- 9223372036854 • MINUTES (MINUTE, MIN): +/- 153722867280 • HOURS (HOUR,HR): +/- 2562047788 • DAYS (DAY): +/- 106751991 The values in parentheses are alternate names for an `interval` unit. When the maximum value for a unit is specified, specifying another unit causes an overflow. Specify each unit only once.
`long`	A signed 64-bit integer. The range of allowed values is -9223372036854775808 to +9223372036854775807 (-2^{63} to 2^{63-1}). To initialize a variable, parameter, or column with a value of -9223372036854775808, specify (-9223372036854775807) -1 to avoid CCL compiler errors.
`money`	A legacy datatype maintained for backward compatibility. It is a signed 64-bit integer that supports 4 digits after the decimal point. Currency symbols and commas are not supported in the input data stream.

Datatype	Description
`money(n)`	A signed 64-bit numerical value that supports varying scale, from 1 to 15 digits after the decimal point. The input data stream supports decimal points, but not currency symbols and commas. The supported range of values change, depending on the specified scale: `money(1)`: -922337203685477580.8 to 922337203685477580.7 `money(2)`: -92233720368547758.08 to 92233720368547758.07 `money(3)`: -9223372036854775.808 to 9223372036854775.807 `money(4)`: -922337203685477.5808 to 922337203685477.5807 `money(5)`: -92233720368547.75808 to 92233720368547.75807 `money(6)`: -92233720368547.75808 to 92233720368547.75807 `money(7)`: -922337203685.4775808 to 922337203685.4775807 `money(8)`: -92233720368.54775808 to 92233720368.54775807 `money(9)`: -9223372036.854775808 to 9223372036.854775807 `money(10)`: -922337203.6854775808 to 922337203.6854775807 `money(11)`: -92233720.36854775808 to 92233720.36854775807 `money(12)`: -9223372.036854775808 to 9223,372.036854775807 `money(13)`: -922337.2036854775808 to 922337.2036854775807 `money(14)`: -92233.72036854775808 to 92233.72036854775807 `money(15)`: -9223.372036854775808 to 9223.372036854775807 To initialize a variable, parameter, or column with a value of -92,233.72036854775807, specify (-9...7) -1 to avoid CCL compiler errors. Specify explicit scale for money constants with `Dn` syntax, where `n` represents the scale. For example, 100.1234567D7, 100.12345D5. Implicit conversion between `money(n)` types is not supported because there is a risk of losing range or scale. Perform the `cast` function to work with money types that have different scale.
`string`	Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, with a size limit of 2 gigabytes. This size limit is reduced proportionally by the size of other content in the row, including the header.
`time`	Stores the time of day as a two-byte field with a range of 00:00:00 to 23:59:59. The default format is HH24:MM:SS.
`msdate`	A timestamp with millisecond precision. The default format is YYYY-MM-DDTHH:MM:SS:SSS.

4.3.10.4 Setting Parameters in Project Configuration

Edit existing parameter definition values and remove deleted parameters.

Context

The list of parameter definitions is automatically populated based on parameters within any CCL documents in the project folder. You can change parameter definition values in the CCR editor. You can remove parameters if the definition has been deleted from the CCL document.

Procedure

1. Select the **Parameters** tab in the Project Configuration editor.
2. To modify a parameter value, click the parameter and change the value in the **Parameter Details** pane.

   > Note
   >
   > You cannot modify the parameter **Name** field.
   >
   > You cannot use functions within parameter value fields. The Project Configuration editor only accepts simple values that adhere to the standards set by smart data streaming datatypes. See *Datatypes* for more information.

3. To remove deleted parameter definitions from the list, select **Remove**.

   > Note
   >
   > A parameter definition marked as (removed) has been deleted from the original CCL file and can be removed from the parameter definition list.

4.3.10.5 Editing Adapter Property Sets in Project Configuration

Use the CCR Project Configuration editor to configure adapter property sets in a project configuration file. Property sets are reusable sets of properties that are stored in the project configuration file. Using an adapter property set also allows you to move adapter configuration properties out of the CCL file and into the CCR file.

Context

Property sets appear in a tree format, and individual property definitions are shown as children to property sets.

Procedure

1. From the SAP HANA Streaming Development perspective, double-click the `.ccr` file to open the CCR Project Configuration editor.
2. Select the **Adapter Properties** tab.
3. (Optional) To create a list of adapter property sets that correspond to the ATTACH ADAPTER statements in the main CCL file for the project, click **Add from CCL**.
4. To create a new adapter property set, click **Add**.
5. In the Property Set Details pane, define a name for the property set.
6. To add a new property to a property set, right-click the set and select **New** > **Property**.

   Note

   You can add as many property items to a property set as required.

7. To configure a property:
   a. In the Property Details pane, define a name for the property.
   b. To define a property as an environment variable:
      - For Windows, use the format %`<environment-variable-name>`%.
      - For Unix, use the format ${`<environment-variable-name>`}.
   c. Enter a value for the property.
8. (Optional) To encrypt the property value:
   a. Select the property value and click **Encrypt**.
   b. Enter the required fields, including Cluster URI and credential fields.
   c. Click **Encrypt**.
      The value, and related fields, are filled with randomized encryption characters.

      Note

      To reset the encryption, click **Encrypt** beside the appropriate field. Change the values, as appropriate, then click **Reset**.

9. To remove items from the All Adapter Properties list:
   - Right-click a property set and select **Remove**, or
   - Right-click a property and select **Delete**.

4.3.10.6 Editing Bindings in Project Configuration

Configure input and output bindings to enable streams or windows in different projects to provide or receive data between one another.

Prerequisites

The streams or windows you want to bind have:

- Compatible schemas
- The same datatype for each field name
- The same column order
- The same number of columns

Context

Configuring bindings directly connects the output of one project to the input of another. Bindings connect projects to one another in the same way that adapters connect projects to outside data sources or destinations.

You can configure bindings from either the source or the destination project—that is, you can choose to publish or to subscribe. An input stream can receive data from different sources through multiple bindings; both input and output streams can provide data to different destinations through multiple bindings.

Bindings can convey data:

- From an output stream or window in the current project to an input stream or window in a remote project. This is called an output binding.
- From a stream or window in a remote project to an input stream or window in the current project. This is called an input binding, and it is the default setting for a binding in the CCR file.
- From an input stream or window in one project to an input stream or window in another project. This is called an input-to-input binding. If you configure an input-to-input binding on the input stream or window that is providing the data, you must select the Output Binding type. By default, an input stream or window assumes that any binding configured on it is an input binding. However, if you configure an input-to-input binding on the input stream or window that is receiving the data, do not set Binding Type to Output Binding. For information on setting the Output parameter in the CCR file, see the *SAP HANA Smart Data Streaming: Developer Guide*.

Binding information is specified in the project configuration (CCR) file so that binding references may be changed at runtime, allowing the project to be used in multiple environments.

Procedure

1. In the CCR Project Configuration editor, select the **Bindings** tab.

2. To add a binding, click **Add**, or to display a list of available streams/windows, click **Discover**.

   You can create multiple bindings on a single stream or window.

3. To configure individual binding settings, use the **Binding Details** pane on the right side of the CCR Project Configuration editor.

Field	Description
Binding Type	Property name as it appears in the ccr file: `Output` Type: `boolean` For most bindings, you need not set this option because it defaults to the correct value. Set this option to **Output Binding** only to configure an input stream or window in this project to send data to an input stream or window in a remote project. When you select **Output Binding** on an input stream, you tell the binding to publish (send data out) to the remote input stream. If you do not check the **Output Binding** box, the binding subscribes to data from the remote input stream because bindings on input streams receive data by default. Note Set this option to **Output Binding** only when you configure a binding on an input stream or window that is providing output. If you configure the binding on the stream or window that is receiving input, do not set this to **Output Binding**. (It is never necessary to select **Output Binding** when you configure a binding on an output stream; output streams can only produce output.)
Binding name	Property name as it appears in the ccr file: `BindingName` Type: `string` (Optional) Apply a name to the binding.
Local stream/window	Property name as it appears in the ccr file: `<Binding name>` Type: `string` Enter the name of the local stream or window (for example, localStream1) or click **Discover** to view and select from a list of streams/windows.
Reconnect Interval (seconds)	Property name as it appears in the ccr file: `ReconnectInterval` Type: `integer` If the connection between the local and remote streams is lost, the project attempts to reconnect at the specified interval. To suppress all reconnection attempts, set Reconnect Interval to 0. Use positive whole number values to set the reconnection interval. Default interval is 5 seconds.
Remote Stream properties	

Field	Description
Cluster	Property name as it appears in the ccr file: `Cluster` Type: `string` Select the cluster that contains the project to bind to.
Remote stream/window	Property name as it appears in the ccr file: `RemoteStream` Type: `string` Enter the name of the remote stream or window (for example, remoteStream1) or click **Discover** to view and select from a list of streams/windows. If you use **Discover**, make sure the cluster and project you are binding to are both running. If they are not, **Discover** cannot find their streams or windows.
Workspace	Property name as it appears in the ccr file: `Workspace` Type: `string` Enter the workspace name (for example, ws1) or click **Discover** to view and select from a list of workspaces. If you use **Discover**, make sure the cluster and project you are binding to are both running. If they are not, **Discover** cannot find their workspaces.
Project	Property name as it appears in the ccr file: `Project` Type: `string` Enter the project to access (for example, project1) or click **Discover** to view and select from a list of projects. If you use **Discover**, make sure the cluster and project you are binding to are both running. If they are not, **Discover** cannot find the project.
Guaranteed Delivery properties	
Enable Guaranteed Delivery	Property name as it appears in the ccr file: `EnableGD` Type: `boolean` Enable GD for a binding to guarantee that if the connection between the binding and the remote stream is severed (by shutting down the project that contains the local stream, for example), all transactions that are supposed to be transmitted through the binding during its downtime are processed once the connection is re-established. **Note** When you enable GD on a binding, make sure: ○ The binding's source data window is running in GD mode or GD mode with checkpoint. ○ The binding's target data window is backed by a log store.

Field	Description
Enable Guaranteed Delivery Cache	Property name as it appears in the ccr file: `EnableGDCache` Type: `string` Enable this binding to cache data. When the source data window is in GD mode with checkpoint, the binding receives checkpoint messages that indicate the last row of data that has been checkpointed by the window. If the binding is enabled for GD caching, it caches incoming transactions until it receives a checkpoint message from the source window. The checkpoint message triggers the binding to send all cached transactions up to the one indicated in the checkpoint message, to the target window. The binding issues a `GD commit` to the source data window after releasing cached data. If GD caching is disabled, streaming ignores checkpoint messages and the binding forwards data based on the **Guaranteed Delivery Batch Size**. Streaming ignores **Enable Guaranteed Delivery Cache** if the source data window is not in GD mode with checkpoint.
Guaranteed Delivery Name	Property name as it appears in the ccr file: `GDName` Type: `string` Supply a unique name for the GD session (subscription) this binding establishes.
Guaranteed Delivery Batch Size	Property name as it appears in the ccr file: `GDBatchSize` Type: `integer` Set this property to customize batch size. The value you enter has to be a positive integer greater than 0. The default value is 1.
Advanced Properties	
Droppable	Property name as it appears in the ccr file: `Droppable` Type: `boolean` (For input bindings only) If the reader cannot keep up, the connection to the bound stream/window is dropped and attempts to reconnect. The default value is false.
Keep Base	Property name as it appears in the ccr file: `KeepBase` Type: `boolean` (For input bindings only) Set this property to true to receive the initial contents of the stream, as well as the updates. The default value is true.

Field	Description
Lossy	Property name as it appears in the ccr file: `Lossy` Type: `boolean` (For input bindings only) If set to true, the binding is lossy and if the binding cannot keep up (meaning when the first project cannot send information to the second project), some data may be lost. If set to false, the binding is not lossy and cannot lose data if it cannot keep up. Setting this property to false may have negative performance impact. The default value is false.
Mirror	Property name as it appears in the ccr file: `Mirror` Type: `boolean` (For input bindings only) If set to true, the input binding stream/window keeps the same data as the remote stream/window to which it is bound and clears the data when the remote stream data is cleared. If set to false, the local binding stream/window does not check the status of the remote stream/window data and always keeps the local input binding stream/window data. The default value is false.
Only Base	Property name as it appears in the ccr file: `OnlyBase` Type: `boolean` (For input and output bindings) Set to true to receive only the initial contents of the stream. The default value is false.
Base Drain Timeout	Property name as it appears in the ccr file: `BaseDrainTimeout` Type: `integer` (For input bindings only) The maximum time, in milliseconds, to receive all base data for the connected stream before the connected remote project forces a disconnection. The default value is 0.
Pulse	Property name as it appears in the ccr file: `Pulse` Type: `integer` (For input bindings only) Specify a period, in seconds, in which the remote stream/window pulses to check for updates. Use a non-zero value to enable this property. The default value is 0.
Queue Size	Property name as it appears in the ccr file: `QueueSize` Type: `integer` (For input bindings only) The maximum number of records to queue up before the binding connection is dropped. The default value is 0.

Field	Description
Query	Property name as it appears in the ccr file: `Query` Type: `string` (For input bindings only) A string which specifies a valid SELECT SQL statement.

4. (Optional) To remove a binding, select it and click **Remove**.

4.3.10.7 Sample Project Configuration File

If you prefer using text editor instead of the Studio Project Configuration editor, use this example to help you build and modify your own project configuration (`.ccr`) files.

In this example, notice that the `.ccr` file is organized in sections according to the preferences being set, including clusters, managers, bindings, parameters, adapters, and project settings. For information on adapters and adapter property sets, see the *SAP HANA Smart Data Streaming: Adapters Guide*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://www.sybase.com/esp/project_config/2010/08/">
  <Runtime>
    <Clusters>
      <!-- We need this only if we have a project/stream binding. -->
      <Cluster name="cluster1" type="local">
        <Username>atest</Username>
        <Password>secret</Password>
      </Cluster>
      <Cluster name="cluster2" type="remote">
        <Username>user2</Username>
        <Password>Pass1234</Password>
        <!-- Managers section is required for a remote cluster. Managers for a
local cluster are retrieved internally from the node. -->
        <Managers>
          <Manager>http://<host>:<RPC port></Manager>
        </Managers>
      </Cluster>
    </Clusters>
    <Bindings>
      <Binding name="stream1">
        <Cluster>cluster1</Cluster>  <!-- this is always needed. -->
        <Workspace>w1</Workspace>
        <Project>p1</Project>
        <BindingName>b1</BindingName>
        <RemoteStream>remote1</RemoteStream>
      </Binding>
      <Binding name="stream2">
        <Cluster>cluster2</Cluster>  <!-- this is always needed -->
        <Workspace>w2</Workspace>
        <Project>p2</Project>
        <BindingName>b2</BindingName>
        <RemoteStream>remote2</RemoteStream>
      </Binding>
      <Binding name="stream3">
        <Cluster>cluster3</Cluster>  <!-- this is always needed -->
        <Workspace>w3</Workspace>
        <Project>p3</Project>
        <BindingName>b3</BindingName>
        <RemoteStream>remote3</RemoteStream>
      </Binding>
```

```
    </Bindings>
    <Parameters>
      <Parameter name="myparam1">foo</Parameter>
      <Parameter name="myparam2">1234</Parameter>
      <Parameter name="myparam3">true</Parameter>
    </Parameters>
    <AdaptersPropertySet>
      <PropertySet name="datalocation1">
        <Property name="myhost1">5555</Property>
      </PropertySet>
      <PropertySet name="datalocation2">
        <Property name="myhost2">6666</Property>
      </PropertySet>
    </AdaptersPropertySet>
  </Runtime>
  <Deployment>
    <Project ha="false">
      <Options>
        <Option name="time-granularity" value="5"/>
        <Option name="debug-level" value="3"/>
        <Option name="java-max-heap" value="512"/>
        <OptionList name="java-classpath">
          <OptionListItem>C:/something/example1.jar</OptionListItem>
          <OptionListItem>C:/something/example2.jar</OptionListItem>
          <OptionListItem>C:/something/</OptionListItem>
        </OptionList>
      </Options>
      <Instances>
        <Instance>
          <Failover enable="false"/>
          <Affinities>
            <Affinity charge="positive" strength="strong" type="controller"
value="myController"/>
          </Affinities>
        </Instance>
      </Instances>
    </Project>
  </Deployment>
</Configuration>
```

If you are ready to deploy projects, see the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for an example of the deployment section of this file.

4.3.11 Compiling a Streaming Project

When you compile a project, smart data streaming produces an executable `.ccx` file from CCL code. This file is necessary to run a project on smart data streaming.

Procedure

1. In the SAP HANA Streaming Development perspective, in the project explorer, expand the tree view to show the `.ccl` file for the project.
2. Select and open the `.ccl` project you are compiling.
3. To compile a project without running it, either to check for errors or just to have an updated `.ccx` file, click **Compile Project**  on the main toolbar, or press **F7**.

4. To compile and run the project, click **Run Streaming Project** .

   The project automatically compiles and runs. The server view in the SAP HANA Streaming Run-Test perspective opens, showing the project connection. A successful connection displays the server streams below the server folder. If the connection is unsuccessful, you see a server connection error dialog.

   > Note
   >
   > You cannot cancel project compilation after it has started. By default, the studio is configured to time out if project compilation takes longer than 5 minutes. If you need more time for a project to compile, consider changing the Operations Timeout setting in the **SAP HANA smart data streaming** Preference settings. For more information, see Editing Preferences for Smart Data Streaming [page 58].

Results

Studio saves all open files belonging to the project, compiles the project, and creates the `.ccx` file (the compiled executable). Compilation errors are displayed in **Problems** or **Console** view in each perspective, depending on the type of error. If you selected **Run Project** , it also runs the compiled project.

> Note
>
> The `.ccx` file is platform-independent. You can migrate, test, develop, and run the file in a studio running on different operating systems. For example, you can compile a project in studio on Unix, and deploy the same file in studio on Windows.

Studio returns an error when a project refers to a schema from an imported file but the project compiles without errors. Refresh the file by closing the project or create the files in the opposite order.

4.3.12 Compiling Projects When Running SAP HANA Studio on Mac OS

Before you can compile and run a streaming project in SAP HANA studio on Mac OS, or on Linux with no smart data streaming server installed, first connect to an SAP HANA smart data streaming server.

Prerequisites

When using smart data streaming in SAP HANA studio on Mac, or on Linux with no smart data streaming server installed, there is no default cluster so you need to update the Default Server URL preference. Before you can update this URL, connect to an SAP HANA smart data streaming cluster in the server view of the SAP HANA Streaming Run-Test perspective. Otherwise, the SAP HANA smart data streaming server will not be available to select. See Connecting to a Cluster [page 66].

Procedure

1. In the SAP HANA Streaming Run-Test perspective of SAP HANA studio, select **Window** > **Preferences**.
2. In the Preferences dialog, select **SAP HANA smart data streaming**.
3. In the **Default Server URL** field, click **Change** and select a server from the Select Default Server URL dialog. Click **OK**.
4. In the Preferences dialog, click **Apply**, then **OK**.
5. (Optional) To compile a project without running it, click **Compile Project** on the main toolbar or press F7.
6. (Optional) To compile a project and run it, click **Run Streaming Project**.

Results

SAP HANA studio is now connected to an SAP HANA smart data streaming server.

4.4 Running and Testing Projects using the SAP HANA Streaming Perspectives

To run a streaming project in SAP HANA studio on Windows or Linux, first connect to an SAP HANA smart data streaming server. Once connected, run the project to upload data from a file, manually enter data, and view streams in that project. Test a streaming project by compiling and running it on a server, accessing and filtering streams, saving and uploading data to the SAP HANA smart data streaming server, and setting project configurations.

Prerequisites

- To run a streaming project, ensure that one or more connected workspaces are available.
- When studio is installed on a separate machine from the streaming server, you cannot run projects on the default local cluster. Instead, connect to the streaming server to run your project.
- Connect to SAP HANA smart data streaming server. See Connecting to a Cluster [page 66] for detailed instructions.

Procedure

1. Select and open the `.ccl` file you want to run.

   If no editors are open, pick a project to run.
2. To run the project, either:
   - Click **Run Project** in the main toolbar (in either the SAP HANA Streaming Development or the SAP HANA Streaming Run-Test perspective) to run the project in the default/most recently used workspace, or,
   - Click the dropdown arrow next to the Run Project tool and choose **Run Project in Workspace**, then select the workspace where this project will run.

   The project runs and shows results in the SAP HANA Streaming Run-Test perspective.

   > **Note**
   >
   > If you have run the project before and encounter an error stating that a failure occurred because the application already exists, the project may still exist on the server in a stopped state. Remove the project from the SAP HANA Streaming Run-Test server view, and restart the project from the SAP HANA Streaming Development perspective.

In this section:

Running a Snapshot SQL Query against a Window [page 117]
In the SQL Query view, run a snapshot SQL query against an output window in a running project, and show the results in the Console.

Playback View [page 117]
The playback view records incoming data to a playback file, and plays the captured data back into a running smart data streaming instance. You can also use it in place of the file upload tool to upload data files at a controlled rate. All date and time stamps within the playback view are assumed to be in UTC.

Debugging [page 122]
The SAP HANA Streaming Run-Test perspective in studio contains two tools for debugging data flow and to assist you in locating and fixing bugs within the project: the debugger, which allows you to set breakpoints, and the event tracer, which shows the impact of each incoming event on all streams and windows of a project.

4.4.1 Caching Project Settings

When running projects within studio, you can save time by caching your run-test views so they open to your previous settings.

Context

Items cached include:

- The list of Stream View stream tabs.
- The current Manual Input Stream.
- The current SQL Query, Playback, Upload, Monitor, Event Tracer projects.

Procedure

1. Click **Windows** > **Preferences** > **Smart Data Streaming Run-Test**.
2. Select **Always start previously running run-test views on project start**.
3. Uncheck **Clear all cached run-test views**.

> Note
>
> Ensure that you clear this option, because if you clear the cache, there will not be anything saved to load. Select this option only if you do not want to cache your run-test settings.

4.4.2 Working in Server View

The server view shows SAP HANA smart data streaming servers available for connecting and running projects.

You can:

- Connect a project.
- Add a new server URL to the list of available connections, remove an existing server, or reconnect all listed servers.
- Show a server in monitor view or event tracer view.
- Load projects into a workspace.

4.4.3 Viewing a Stream

Stream view shows all of the events of an output stream and all of the retained events in an output window for the running project.

Procedure

1. In the SAP HANA Streaming Run-Test perspective, select the stream or window from the server view.
2. Right-click the output stream or window, and select **Show In** > **StreamViewer** (or **New StreamViewer**).
   A tab opens in the stream view showing all new events. If you selected a window, all retained rows currently in the window are displayed.
3. To manipulate your subscription list, or individual stream subscriptions, select the subscription to edit and choose one of these buttons at the top of the stream view:
   - **Close Subscription URL** disconnects and closes the stream view.
   - **Clear** clears contents and pauses the subscription.
   - **Show Current Subscription in new View** shows the publish date of the stream (if available).
   - **Set StreamViewer number of rows displayed** lets you choose the number of rows to show in the stream view. The number can be between 1 and 1000000. The default value is 25.
4. (Optional) To save data from the stream view, click **Clipboard Copy**.

4.4.4 Specifying a Pulse Rate

You can specify how often you see changes and updates in a stream. When a data stream contains few items with a high volume of changes, you can set a pulse rate so that changes are delivered periodically, in optimally coalesced blocks.

Context

A stream containing three ticker symbols, for example, may generate thousands of updates every second. You can set the pulse period to control the frequency at which you receive updates when viewing the stream. If you set the pulse to refresh every 5 seconds, the subscription then delivers, at most, one updated record for each of the three symbols every five seconds.

There are two preferences that control the subscription feature in studio:

- **StreamViewer pulsed subscribe interval**
- **Other pulsed subscribe interval**

Both preferences are measured in seconds. If either of these preferences is set to 0, then studio does not perform a pulsed subscription on the related stream.

> **Note**
>
> If you have a small data set and you set the pulse to refresh frequently, such as once every 1 or 2 seconds, the Stream View may be empty for some streams because there are no new updates

Go through the following steps to change the default settings:

Procedure

1. Choose **Window** > **Preferences**.
2. In the left pane, expand **Smart Data Streaming** > **Run-Test**.
3. Enter new values for **StreamViewer pulsed subscribe interval** or **Other pulsed subscribe interval** or both.
4. Click **Apply**.
5. Click **OK** to close the dialog.

4.4.5 Uploading Data to the SAP HANA Smart Data Streaming Server

Use the File Upload tool to load event data from files into a running project. Normally, you use this tool to test projects. Date and time stamps in data loaded through the File Upload tool are assumed to be in the local timezone.

Prerequisites

Ensure that the project is running.

Procedure

1. In the SAP HANA Streaming Run-Test perspective, select the **File Upload** view in the lower-left pane.

   > Note
   >
   > The File Upload tool uploads the data file as fast as possible. For playing back data at controlled rates, use the Playback tool.

2. Click **Select Project** in the toolbar in the upper right corner of the File Upload view.
3. Select the project to which you want data uploaded, and click **OK**.
4. Click **Browse** to open the file choice dialog and navigate to the input file to upload.
5. Select one or more files to upload:
   - The smart data streaming server supports smart data streaming binary (`.bin`), smart data streaming XML (`.xml`), and comma-separated values and text (`.csv` or `.txt`) files. Regardless of file type, each record in the file has to start with the input stream or window name in the first field, followed by the opcode in the second field, then the actual contents of the record in the remaining fields.
   - Use the format `yyyy-MM-'dd'T'HH:mm:ss` when uploading any `SecondDate` data from a file. For example, `TradeTime="2000-05-04T12:00:00"`.
   - Use the format `yyyy-MM-'dd'T'HH:mm:ss.SSS` when uploading any `MsDate` data from a file. For example, `Tradetime="2000-05-04T12:00:00.000"`.
6. Click **Upload**. A progress bar tracks the upload status.

   The File Upload view allows you to perform these additional actions:

UI control	Action
Remove File	Discard a previously selected file from the Input File(s) menu.
Cancel	Cancel a file upload currently in progress.

UI control	Action
	Note Any data sent before the upload is cancelled is still processed.
Use Transaction	Process multiple records as a single transaction. If **Record Buffer** is specified, group that many records in each transaction. If not, process the entire file as one transaction.
Record Buffer	Specify the number of records to group together and process in a single transaction.

4.4.6 Manually Entering Data to a Stream

Manually create and publish an event as input to a stream or window. By default, date and time stamps in data loaded through the Manual Input tool are assumed to be in the local timezone. You can change this setting to use Universal Coordinated Time (UTC) through your Studio preferences.

Context

Manually publishing input events to a project is useful when testing a project.

Procedure

1. In the SAP HANA Streaming Run-Test perspective, select the **Manual Input** view in the lower-left pane.
2. Click **Select Stream** ( ) in the toolbar in the upper right corner of the Manual Input view.
3. In the Select Stream dialog, select the stream and click **OK**.
4. Edit available data columns as desired.
5. To edit more than one row of the data, select **Edit Multiple Rows** ( ) and choose the rows to modify.
6. If you are publishing to a window, indicate the opcode by selecting one of the data events. If you are publishing to a stream, only insert events are supported.
7. (Optional) Select **Use Current Date** to change the value of any bigdatetime or date object in the manual input view to the present date.
8. Click **Publish** to send the event to the project.

Related Information

Manual Input Settings [page 59]

4.4.7 Monitoring Project Performance

The Monitor View shows visual indicators of queue size, throughput, and CPU use for each stream and window (including local streams and windows) in a project.

Each node corresponds to a stream in the model with the lines outlining the path the data flows through. The color of each node represents either QueueDepth or Rows Processed (/sec), depending on your specifications.

For example, if you select the **Color Queue Depth** option, the (Red) Range >= field defaults to 125, and the (Yellow) Range >= field defaults to 20. This means:

- If the queue depth of the stream node is greater than or equal to 125, the node is red.
- If the queue depth of the stream node is between 20 and 124, the node is yellow.
- If the queue depth of the stream node is less than 20, the node is green.
- If the nodes remain white, it indicates that the monitor is not receiving data from the stream processor.

The Monitor View depicts CPU utilization as a black pie wedge in the ellipses of the node. Based on the options chosen, the remainder of the ellipses are red, yellow or green. A fully black node represents 100 percent CPU use, based on a single CPU core. With multicore or multiprocessor environments, a fully black node may be greater than 100 percent.

You can look at a specific node's performance statistics by moving your cursor over the node in the diagram.

> **i Note**
>
> Monitoring data is available only if the time-granularity option (otherwise known as the performance monitor refresh interval) is enabled. The time-granularity option specifies, in seconds, how often the set of performance records — one per stream and one per gateway connection — is obtained from the running smart data streaming project.

In this section:

4.4.7.1 Running the Performance Monitor

View visual indicators of queue size and CPU usage for each stream and window.

Prerequisites

The project you wish to monitor is running.

Context

You can specify a delay by changing the performance timer interval.

Procedure

1. In the studio SAP HANA Streaming Run-Test perspective, select the server view.
2. Right-click on a running project and select **Project Properties**.
3. In the dialog, set **Performance Monitoring Refresh Interval** to a number greater than 0.
   The default setting is 5. The performance monitor will not run if the refresh interval is set to 0.
4. In the monitor view, click **Select Running Project** ().
5. Click **OK**.
6. Select **QueueDepth** or **Rows Processed** to specify how to color each node in the performance diagram. For either option:
   - Type in a number or use the arrow buttons in the (Red) Range >= field to select the range to create a red node.
   - Type in a number or use the arrow buttons in the (Yellow) Range >= field to select the range to create a yellow node.

   > **Note**
   > Nodes are green when they fall within a range that is not in either the (Red) Range >= or the (Yellow) Range >=.
7. Click **Zoom In** or **Zoom Out** to see a larger or smaller view of the diagram.

4.4.7.2 Saving a Performance Diagram as an Image

Save a performance diagram.

Context

You can modify the performance diagram properties using the Monitor window in the SAP HANA Streaming Run-Test perspective. The diagram appears in the Event Tracer window, and can be saved only from that window.

Procedure

1. In the SAP HANA Streaming Run-Test perspective, select the **Event Tracer** view.
2. Click **Save**.
3. Enter a file name and save location. Click **Save**.
   The file is saved as a JPEG image in the location you specified.

4.4.8 Running a Snapshot SQL Query against a Window

In the SQL Query view, run a snapshot SQL query against an output window in a running project, and show the results in the Console.

Procedure

1. In the SAP HANA Streaming Run-Test perspective, select the **SQL Query** view in the lower-left pane.
2. Click **Select Project** .
3. Select the project and window to query, and click **OK**.
4. Enter your query.
   For example, `Select * from <stream>`.
5. Click **Execute**.
   The results are displayed in the Console.

4.4.9 Playback View

The playback view records incoming data to a playback file, and plays the captured data back into a running smart data streaming instance. You can also use it in place of the file upload tool to upload data files at a controlled rate. All date and time stamps within the playback view are assumed to be in UTC.

Playback View Options

Feature	Description
Select playback file	Select the file format to use with the smart data streaming recorder.
Start playback	Starts playing the current playback file.
Stop playback	Stops playback or record, closes the associated file and closes the associated playback or record context.

Feature	Description
Start Recording	Prompts the user to select the file in which to store recorded data and starts the smart data streaming recorder.
At Timestamp rate	This slider is used during playback to vary the rate of playback
XML/CSV datemask	Applies a datetime mask to read data from the source file. The default datetime format is UTC, or %Y-%m-%dT%H:%M:%S. This option cannot change delimiters, which are the characters that separate each value: "-", "T", ":").

Playback Mode Options

Feature	Description
Full rate	**Full rate** indicates that the speed of playback is not imposed by the studio. **Full rate** is dependent on factors such as the computer that is running studio, or network latency.

Feature	Description
Timestamp column	The **timestamp column** option tells the recorded file to play back using the timing rate information from a specified column. During playback, timestamps determine the time interval between records. To change the project runtime to the time the data was recorded: 1. Select **timestamp column**. 2. Check **Use Recorded time**. 3. Enter a column name in the **timestamp column** field. To return to current time, restart the project. The **timestamp column** field supports these datetime datatypes: • `BIGDATETIME` • `SECONDDATE` • `TIME` • `INTERVAL` • `INTEGER` • `LONG` When entering a datemask in the **XML/CSV datemask** field, these datatypes have a default format: • `BIGDATETIME`: YYYY-MM-DDTHH:MM:SS:SSSSSS • `SECONDDATE`: YYYY-MM-DDTHH:MM:SS • `TIME`: HH24:MM:SS **Note** These formats cannot change delimiters. These datatypes do not support a datemask: • `INTERVAL` • `INTEGER` • `LONG`
Rec/ms	The records-per-millisecond (**rec/ms**) mode lets playback occur at a records-per-millisecond rate. The option allows you to set an initial rec/ms rate, which you can modify using the **At Timestamp rate** slider tool.

Supported File Extensions and Operation Codes

The studio recorder supports these file formats:

- `.xml` (smart data streaming XML)
- `.csv` (comma-separated values)
- `.bin` (smart data streaming Binary)

- `.rec` (studio recorded file)

Regardless of file type, each record in the file has to start with the input stream or window name in the first field, followed by the opcode in the second field, then the actual contents of the record in the remaining fields. The default date/time format for all supported date/time datatypes is `%Y-%m-%dT%H:%M:%S`. For example, `2011-06-30T17:00:00.000`.

Supported operation codes, and their abbreviated forms, are:

- INSERT: `I` or `i`
- UPDATE: `U` or `u`
- DELETE: `D` or `d`
- SAFEDELETE: `S` or `s`
- UPSERT: `P` or `p`

Additional Formatting Rules for CSV and XML Files

For CSV files, the value order has to correspond to the schema definition of the input stream or window. For example:

```
isSensorStream,i,2011-06-30T17:00:00.000,34.614,-111.843,Tower,Camp Verde
T11,1,96.8
```

For CSV files, null values are only allowed if the correct number of fields are represented and delimited by commas (or whatever single-character delimiter is used from the ASCII printable character set). For example:

```
isSensorStream,i,2011-06-30T17:00:00.000,,,,Camp Verde T11,,96.8
```

For XML files, the columns can be in any order as long as the column names correspond to the schema names. You can omit any fields rather than leaving them empty (null value). For example:

```
<Positions ESP_OPS="i" BookId="Book1" Symbol="MSFT" SharesHeld="3000"
Basis="528400"/>
```

Binary files recorded in previous releases cannot be played back until they are converted to the new binary format using `streamingconvert`. See *streamingconvert* in the *SAP HANA Smart Data Streaming: Utilities Guide*.

SAP HANA smart data streaming records in `.rec` format, preserving the original timing of the incoming data.

> **Note**
>
> Binary messages are architecture-dependent. Binary messages created in a big-endian machine cannot be loaded into the smart data streaming server running in a little-endian machine, and vice-versa.

In this section:

Recording Incoming Data in a Playback File [page 121]
Record incoming data that is flowing into smart data streaming to a playback file that you can save and view at a later time.

Playing Recorded Data [page 122]
View and play previously recorded data in a running smart data streaming instance.

Related Information

SAP HANA Smart Data Streaming: Utilities Guide

4.4.9.1 Recording Incoming Data in a Playback File

Record incoming data that is flowing into smart data streaming to a playback file that you can save and view at a later time.

Prerequisites

Connect to an SAP HANA smart data streaming server. Have your streams and windows visible in the Stream View.

Procedure

1. Open the SAP HANA Streaming Run-Test perspective.
2. In the Playback view, click **Select Project** .
3. Select the project you want to record. Click **OK**.
4. Click the **Record** icon.
5. Select the streams and windows to record, or click **Select All** to record all streams and windows in the project. Click **OK**.
6. Select or create a file in which to save the recording. Click **OK**.
7. Send data to your selected streams using either:
    - The Manual Input view to input data and publish to your streams, or,
    - **File Upload** to retrieve an existing data file and publish to your streams.
8. Click **Stop** to stop recording.

4.4.9.2 Playing Recorded Data

View and play previously recorded data in a running smart data streaming instance.

Context

> **Note**
> You may select Playback view options before or after you select a file for playback.

Procedure

1. Click **Playback File** .
2. Browse for and select the file you want to play back.
   The playback file is added to the Playback File History. You can also play back a file registered in the history. Double-click a history entry to activate it for playback.
   > **Note**
   > You can delete an item from the history using the either the **Remove** button or **Delete** key. Modifications to the playback history are permanent.
3. Click **Play** to begin playback.
   By default, the data appears in the Stream View at the rate it was recorded.

4.4.10 Debugging

The SAP HANA Streaming Run-Test perspective in studio contains two tools for debugging data flow and to assist you in locating and fixing bugs within the project: the debugger, which allows you to set breakpoints, and the event tracer, which shows the impact of each incoming event on all streams and windows of a project.

Use the debugging tools during project development, not while smart data streaming is in production mode. Debugging features are normally disabled. Place the system in Trace mode before using the debugging features.

Studio offers an extensive suite of tools for debugging projects, but you can debug from the command line as well. See the *SAP HANA Smart Data Streaming: Utilities Guide*.

In this section:

Event Tracer View [page 123]
The event tracer is a tool for debugging data flow. It shows the impact an event has on each stream and window of the project.

[Debugging with Breakpoints and Watch Variables [page 125]](#)
Use studio to control a running project by enabling tracing, pausing, resuming, and stepping of data flow through smart data streaming streams. You can also create breakpoints and watch variables on a running application.

Related Information

SAP HANA Smart Data Streaming: Utilities Guide

4.4.10.1 Event Tracer View

The event tracer is a tool for debugging data flow. It shows the impact an event has on each stream and window of the project.

The event tracer view shows the transaction flow through the model and lets you view data in each node (stream or window). The nodes depicted in the event tracer view are drawn as a data flow, depicting the relationships between the nodes. The following table describes the function of each available button in the event tracer view.

Button	Function
Select Running Project	Presents a list of running projects available to monitor from studio.
Layout TopDown	Arranges shapes vertically for a top-to-bottom data flow.
Layout Left to Right	Arranges shapes horizontally for a left-to-right data flow.
Save	Saves the image as a JPG file.
Zoom In	Enlarges the size of the image.
Zoom Out	Reduces the size of the image.
Zoom Page	Restores the size of the image to its original size.
Print Performance Data to Console	Prints the collected data to the console.
Close Subscription	Closes the subscription and clears the view.
Show Current Subscription in New View	Displays the current subscription in a separate view.
Fit Shape Ids	Expands a shape to show the name of the stream or window.
Initialize With Base Data	Sends all event data from smart data streaming through the event tracer.

In this section:

4.4.10.1.1 Tracing Data Flow in the Event Tracer

Run the event tracer to trace data flow.

Prerequisites

The smart data streaming server is running.

Procedure

1. In studio, in the SAP HANA Streaming Run-Test perspective:

Method	Procedure
Event Tracer	1. Click the event tracer view. 2. Click **Select Running Project** ( ) to show running projects that contain streams or windows. 3. Select a running project for the event tracer. 4. Click **OK**.
Server View	1. Select the server view. 2. To refresh the server view, click **Reconnect All**. 3. Select a running project that contains streams. 4. Right-click the project node, and select **Show in** > **Event Tracer View**.

   The nodes depicted in the viewer are drawn as a data flow. As a transaction is processed by each node, the color of the node changes to reflect the type of transaction.

2. Double-click a node to show the corresponding stream's data in the console view.
3. Load test data to view the impact on each stream in the event tracer. Do one of the following:
   - Click the **Upload File** tab in the toolbar below the Activate Project pane to upload data from a file.
   - In the manual input view, manually enter individual transactions by clicking the **Select Stream** icon. Select a stream. To confirm, click **OK**.

Results

The shapes in the event tracer view change color.

4.4.10.1.2 Viewing the Topology Stream

The Topology Stream constructs the data-flow diagram, where relationships between the nodes of a project are represented as line segments.

Procedure

1. In the SAP HANA Streaming Run-Test perspective, select **Event Tracer** view.
2. Click **Select Running Project**. Select the desired project, and click **OK**.
3. To view the entire diagram, select **Layout top down** or **Layout left to right**.
4. To view a particular node, select the section of the data-flow diagram that contains the desired stream.

4.4.10.2 Debugging with Breakpoints and Watch Variables

Use studio to control a running project by enabling tracing, pausing, resuming, and stepping of data flow through smart data streaming streams. You can also create breakpoints and watch variables on a running application.

Breakpoints are locations in a stream or window's input or output that stop the flow of data in the smart data streaming model. A watch variable inspects the data. The following table describes the function of each studio breakpoint button:

Button	Function
Trace On	Instructs smart data streaming to begin tracing (debugging). Set this parameter to use the smart data streaming breakpoint APIs.
Trace Off	Stops tracing (debugging).
Step Project	Steps through the running smart data streaming project.
Pause Project	Pauses playback for projects recorded as .rec files. **Note** When the project is paused, the records from Manual Input and File Upload cannot be updated to streams until the project is resumed.
Enable All Breakpoints	Enables all breakpoints in the list.
Disable All Breakpoints	Disables all breakpoints in the list.
Insert Breakpoint	Inserts a breakpoint item into the watch table.
Insert Watch	Inserts a watch item into the watch table.
Print Breakpoint Data to Console	Prints the breakpoint and pause state data for the current smart data streaming to the console.

Breakpoint Commands

The following breakpoint commands initiate long-running operations. Each of these can be cancelled before completion by clicking **Cancel Current Step**.

Button	Function
Step Quiesce from Base	Automatically steps all the derived (non-base) streams until their input queues are empty.
Step Quiesce	Automatically steps the stream and all its direct and indirect descendants until all of them are quiesced.
Step Transaction	Automatically steps until the end of transaction.
Step Quiesce Downstream	Steps the descendants of the stream but not the stream itself.

> i Note
>
> Breakpoints and watch variables persist to the workspace.

In this section:

4.4.10.2.1 Breakpoints

You can insert a breakpoint for any stream in the project.

Breakpoint types include:

Local	breaks on input to the stream.
Input	breaks on a specific input stream to a stream (only Flex, join, and union can have multiple input streams).
Output	breaks when data is output from the stream.

A breakpoint can be associated with a counter (enableEvery). When a counter (n) is associated with a breakpoint, the breakpoint triggers after n events flow through the breakpoint. The counter is then reset to zero.

In this section:

Parent topic: Debugging with Breakpoints and Watch Variables [page 125]

Related Information

Watch Variables [page 128]
Pausing SAP HANA Smart Data Streaming [page 129]
Stepping Through a Project in Studio [page 130]

4.4.10.2.1.1 Adding Breakpoints

Add breakpoints to streams in smart data streaming.

Prerequisites

- The Debugger view of the SAP HANA Streaming Run-Test perspective in studio is open.
- Trace mode is enabled.

Procedure

1. Click **Trace On**.
2. Click **Insert Breakpoint** ( ).
3. Select the stream where you want to set a breakpoint.
4. Select the type of stream.
5. Specify when the breakpoint should trigger by entering a value in the **enableEvery** field.
6. Click **Add**.

   The selected stream appears in the table within the Insert Breakpoint dialog box.
7. Click **OK**.

   The breakpoint appears in the Debugger view within the Breakpoint table.
8. Right-click the breakpoint and select an option:
   - **Enable Breakpoint**
   - **Disable Breakpoint**

- **Remove Breakpoint**

9. (Optional) To enable or disable all breakpoints, select either **Enable All Breakpoints** or **Disable All Breakpoints**.
10. (Optional) To remove all breakpoints, right-click within the Breakpoints table and select **Remove All Breakpoints**.
11. Click **Trace Off** to run smart data streaming.

4.4.10.2.2 Watch Variables

Insert watch variables into the watch table of the Breakpoints view in the Debugger to inspect data as it flows through the project.

A watch corresponds to:

- Current input of a stream
- Current output of a stream
- Queue of a stream
- Transaction input of a stream
- Transaction output of a stream
- Output history of a stream
- Input history of a stream

Add the watches you want to monitor to the watch table before running smart data streaming. When smart data streaming runs, the watch table is updated dynamically as run-control events (run, step, pause) are sent through smart data streaming.

In this section:

Parent topic: Debugging with Breakpoints and Watch Variables [page 125]

Related Information

Breakpoints [page 126]
Pausing SAP HANA Smart Data Streaming [page 129]
Stepping Through a Project in Studio [page 130]

4.4.10.2.2.1 Adding Watch Variables

Add a watch element to a breakpoint.

Prerequisites

- The Debugger view of the SAP HANA Streaming Run-Test perspective in studio is open.
- Trace mode is enabled.

Procedure

1. Click **Trace On**.
2. Right-click in the Watch table.
3. Select **Add Watch**.
4. Select a stream from the Watch Choices box.
5. Select the type of watch you want to set up on that stream.
6. Click **Add**.
   The watch appears in the table at the bottom of the dialog box.
7. Click **OK**.
   The watch appears in the Watch table in the Debugger view.
8. To remove watches, right-click within the Watch table and select, either:
   - **Remove Watch** to remove a single select watch variable, or,
   - **Remove All Watches** to remove all watch variables.

4.4.10.2.3 Pausing SAP HANA Smart Data Streaming

Pause SAP HANA smart data streaming while playing back projects with `.rec` file types.

Prerequisites

- The Debugger view of the SAP HANA Streaming Run-Test perspective is open.
- Trace mode is enabled.

Procedure

1. In the Debugger, click **Pause Project** ( ).
2. To resume smart data streaming, click **Resume Project,** or click **Trace Off** to close the debugger.

Task overview: Debugging with Breakpoints and Watch Variables [page 125]

Related Information

Breakpoints [page 126]
Watch Variables [page 128]
Stepping Through a Project in Studio [page 130]

4.4.10.2.4 Stepping Through a Project in Studio

Single-step through a project in SAP HANA smart data streaming.

Prerequisites

- The Debugger view of the SAP HANA Streaming Run-Test perspective is open.
- The project is paused.

Procedure

1. In the Debugger view, click **Step Project** () to perform the next step in the project.
2. Click **Cancel Current Step** to terminate the action.

Task overview: Debugging with Breakpoints and Watch Variables [page 125]

Related Information

Breakpoints [page 126]
Watch Variables [page 128]

Pausing SAP HANA Smart Data Streaming [page 129]

5 Beyond the Basics

Use advanced CCL techniques, such as declare blocks, modules, and Flex operators, to develop sophisticated and complex projects.

In this section:

5.1 Keyed Streams

Keyed streams save resources by letting you pass insert, update, and delete events through a project without storing the events in memory. Keyed streams also let your project perform certain relational operations, including joins, computes, and filters, without storing the data in memory.

To create a keyed stream, define a primary key for a stream. Inserts, updates, and deletes are assumed to be related to this primary key.

Like other streams, keyed streams can be either input or derived.

A keyed stream:

- Supports a primary key but does not ensure that it is unique.
- Rejects events that have a null primary key value.
- Propagates insert, update, and delete opcodes as is, without modifying or validating them. Ensure that you validate these inserts, updates, and deletes elsewhere in the project.
- Does not detect duplicate inserts, bad updates, or bad deletes.
- Rejects events with upsert or safedelete opcodes.
- Treats all events as inserts for processing purposes, though they may contain different opcodes.

Supported Operations and Features

When a keyed stream is the target for the result of the operation, it supports:

- Inputs
- Computes
- Unions
- Pattern Matching
- Filters (see Filters [page 134] for details)
- Simple joins (see Joins [page 135] for details)
- Flex operations (see Inputs and Outputs [page 134] for details)

Keyed streams support guaranteed delivery.

Unsupported Operations and Features

When the keyed stream is the target for the result of the operation, it does not support:

- Aggregations.
- Joins in which the only inputs are windows. You can get around this by performing a window-window join first, then feeding the results into a keyed stream.

Keyed streams reject:

- Upserts

- Safedeletes
- Any record with a primary key column that has a null value

For additional restrictions, see Inputs and Outputs [page 134], Joins [page 135], and Filters [page 134].

Inputs and Outputs

Keyed streams can send to and receive from other streams (including other keyed streams), Flex operators, and windows. They can serve as inputs to relational operations like joins, aggregations, and computes, or as outputs to relational operations.

Exceptions and considerations:

- A keyed stream cannot feed a window. Add a KEEP clause to the keyed stream to turn it into an unnamed window, which allows it to use memory and retain its state.
  - If you use a KEEP ALL clause, the unnamed window validates inserts, updates, and deletes.
  - If you use a KEEP clause with any other retention policy, the unnamed window treats updates as upserts and deletes as safedeletes. The unnamed window traps duplicate inserts unless the retention policy has allowed the original insert to be purged.
- When a stream feeds a keyed stream, the keyed stream produces inserts. When a keyed stream feeds a keyless stream, the stream follows its semantics of converting updates to inserts and silently dropping deletes.
- When a window feeds a keyed stream, the keyed stream outputs the inserts, updates, and deletes it receives with no changes.
- When a keyed stream feeds a Flex operator, the CCLScript code does not have access to the old record in the case of an update. The old record is always null.
- When a Flex operator feeds a keyed stream, the CCLScript code can generate only insert, update, and delete opcodes. Upsert and safedelete opcodes are not allowed.
- When a keyed stream feeds an event cache, the coalesce option is limited to the case when the records are coalesced on the key field.

Filters

When you use a WHERE clause, filter on columns that have values that do not change between an insert event and subsequent update and delete events. If the columns change, related events can be lost; for example, downstream elements might receive update or delete events without the insert that provided the data being updated or deleted, or fail to receive a delete for a previous insert.

In this example, we create two very similar elements, a keyed stream named `KS1`, and a window named `W1`:

```
CREATE OUTPUT STREAM KS1 PRIMARY KEY (Key1) AS SELECT In1.Key1,
In.Val1, In1.Val2 FROM In1 WHERE In1.Val1 > 10;
CREATE OUTPUT WINDOW W1 PRIMARY KEY (Key1) AS SELECT In1.Key1,
In.Val1, In1.Val2 FROM In1 WHERE In1.Val1 > 10;
```

Suppose `In1` sends this data:

```
<In1 ESP_OPS="I" Key1="1" Val1="5" Val2="abcd"/>
```

```
<In1 ESP_OPS="u" Key1="1" Val1="15" Val2="abcd"/>
<In1 ESP_OPS="d" Key1="1" Val1="6" Val2="abcd"/>
```

Keyed stream `KS1` and window `W1` produce different output:

```
<KS1 ESP_OPS="u" Key1="1" Val1="15" Val2="abcd"/>
<W1 ESP_OPS="i" Key1="1" Val1="15" Val2="abcd"/>
<W1 ESP_OPS="d" Key1="1" Val1="15" Val2="abcd"/>
```

Rather than filtering on `Val1`, which changes, filter on `Val2`, which does not. This approach provides more predictable results.

Joins

When a keyed stream is the target of an inner join, ensure that the columns on which the join is performed do not change across an insert event and related update and delete events that follow. If the columns change, related events can be lost; the keyed stream may send update or delete events without the insert that provided the data being updated or deleted, or fail to send a delete for a previous insert.

Keyed streams are stateless except when performing a join with a window. In this type of join, the keyed stream uses memory to store a reference to the records in the window.

Restrictions on joins:

- A keyed stream can only be an outer member of an outer join (inner joins are supported).
- A keyed stream may not participate in a full join.
- When you join a keyed stream with a window, only events that arrive in the keyed stream trigger a join; changes to the window do not.
- A keyed stream cannot be the target of a join when all inputs to the join are windows (named or unnamed).

5.2 Implicit Columns

All streams, windows, and keyed streams use three implicit columns called ROWID, ROWTIME, and BIGROWTIME.

Column	Datatype	Description
ROWID	long	Provides a unique row identification number for each row of incoming data.
ROWTIME	seconddate	Provides the last modification time as a date with second precision.
BIGROWTIME	bigdatetime	Provides the last modification time of the row with microsecond precision. You can perform filters and selections based on these columns, like filtering out all of those data rows that occur outside of business hours.

You can refer to these implicit columns just like any explicit column (for example, using the `stream.column` convention).

5.3 Adding a Binding to a Streaming Project

Add a binding into your project using the visual editor in the SAP HANA Streaming Development perspective.

Prerequisites

The two streaming projects that you wish to bind together are ready for editing.

Context

There are three ways to add a binding to your project.

Add a binding by dragging a stream or window from the project explorer

Procedure

1. Open the project to which you want to add a binding in the visual editor.
2. In your project explorer, expand the folder of the second project to bind to the first project.
3. Click the output stream or window to bind to the first project, and drag-and-drop it into the first project's diagram.
    - A new input stream or window is automatically created and added to the diagram of the first project. The new stream or window is named using the format <First-Project-Name>_<Stream-Or-Window-Name> and has an inline schema that is identical to the schema of the stream/window from the second project.
    - An icon appears to the left side of the stream or window that contains the binding, signifying either an input or an output binding.
    - A new tab for the Project Configuration editor opens within the visual editor.
    - A new binding is created and automatically added to the Project Configuration (ccr) file, which contains values for the **Binding name**, **Local stream/window name**, **Remote stream**, **Workspace**, and **Project** properties.
4. Use the Project Configuration editor to specify a value for the **Cluster** property, and optionally, values for other properties.
5. Save your changes to the project, as well as the project configuration file.

Add a binding from a stream or window shape within a diagram

Procedure

1. Open the project to which you want to add a binding in the visual editor.
2. In the project diagram, click the binding icon in the stream/window shape to bind to another stream or window.
   A new **Select Binding Target** window opens.
3. Select the stream/window to bind to the first stream or window, and click **OK**.
   - A new tab for the Project Configuration editor opens within the visual editor.
   - An icon appears to the left side of the stream or window that contains the binding, signifying either an input or an output binding.
   - A new binding is created and automatically added to the Project Configuration (ccr) file, which contains values for the **Binding name**, **Local stream/window name**, **Remote stream**, **Workspace**, and **Project** properties.
4. Use the Project Configuration editor to specify a value for the **Cluster** property, and optionally, values for other properties.
5. Save your changes to the project, as well as the project configuration file.

Add a binding by clicking within a diagram

Procedure

1. Open the project to which you want to add the binding in the visual editor.
2. Right-click any blank space in the diagram, and select **Add** > **Binding (in CCR)**.
   A new **Binding (in CCR)** window opens.
3. Select a stream or window for the **Local stream/window** field and a remote stream or window for the **Remote stream** field.
4. Click **OK**.
   - A new tab for the Project Configuration editor opens within the visual editor.
   - An icon appears to the left side of the stream or window that contains the binding, signifying either an input or an output binding.
   - A new binding is created and automatically added to the Project Configuration (ccr) file, which contains values for the **Binding name**, **Local stream/window name**, **Remote stream**, **Workspace**, and **Project** properties.
5. Use the Project Configuration editor to specify a value for the **Cluster** property, and optionally, values for other properties.
6. Save your changes to the project, as well as the project configuration file.

5.4 Splitting Inputs into Multiple Outputs

The splitter construct is a multi-way filter that sends data to different target streams depending on the filter condition. It works similar to the ANSI 'case' statement.

Context

You can create a splitter to provide an operator that can split an input into multiple outputs.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor palette, under **Streams and Windows**, select **Splitter**.
3. Select a location in the diagram and click to add the shape.
4. To set the name of the splitter, either:
    - Click to edit the shape name, or, press **F2**.
    - In verbose mode, click the **Edit** icon next to the name.
5. (Optional) Click to make it an output (instead of local) if you want the splitter outputs to be visible via subscription in the runtime model.
6. Connect the splitter to a single input stream or a window.
7. (Optional) Add or remove `Column Expressions` for the splitter.
8. Create the splitter logic using `Add When` and `Add Else`. This creates the splitter output elements.
9. (Optional) Connect the splitter output elements of the splitter to other streams or windows.

5.5 Declare Blocks

Declare blocks enable you to include elements of functional programming, such as variables, parameters, typedefs, and function definitions in CCL data models.

CCL supports global and local declare blocks.

Global declare blocks	Available to an entire project; however, you can also set individual global declare blocks for each module.

> **Note**
>
> Global declare blocks are merged together if more are imported from other CCL files. Only one is allowed per project.

Local declare blocks Declared in CREATE statements, are available only in the SELECT clause of the stream or window in which they are declared.

> **Note**
>
> The variables and functions defined in a local declare block are only available in the SELECT clause and anywhere inside the Flex operator.

CCL variables allow for the storage of values that may change during the execution of the model. Variables are defined in the declare block using the CCLScript syntax.

CCL typedefs are user-defined datatypes and can also be used to create an alias for a standard datatype. Use typedef to shorten long type names. Once a typedef has been defined in the declare block, use it instead of the datatype in all CCLScript statements, and throughout the project.

CCL parameters are constants for which you can set the value at the model's runtime. Use these parameters instead of literal values in a project to allow behavior changes at runtime, such as window retention policies, store sizes, and other similar changes that can be easily modified at runtime without changing the project. Define CCL parameters in a global declare block, and initialize them in a project configuration file. You can also set a default value for the parameter in its declaration, so that initialization at server start-up is optional.

You can create CCLScript functions in a declare block to allow for operations that are more easily handled using a procedural approach. Call these CCLScript functions from stream queries and other functions throughout the project.

5.5.1 Typedefs

Use typedefs to declare new names for existing datatypes.

Syntax

```
typedef existingdatatypeName newdatatypeName;
```

Components

Component	Description
existingdatatypeName	The original datatype.
newdatatypeName	The new name for the datatype.

Usage

Typedefs allow you to give new names for existing datatypes, which you can use to define new variables and parameters, and specify the return type of functions. Declare typedefs in declare blocks, UDFs and inside Flex procedures. The types declared in typedefs must resolve to simple types.

> **Note**
>
> For unsupported datatypes, use a typedef in a declare block to create an alias for a supported datatype.

Example

This example declares euros to be another name for the money(2) datatype:

```
typedef money(2) euros;
```

Once you have defined the euro typedef, use:

```
euros price := 10.80d2;
```

The example is the same as:

```
money(2) price := 10.80d2;
```

5.5.2 Parameters

Parameters are constants that you set during project setup using the server-command name or the project configuration file.

Syntax

```
parameter typeName parameterName1 [:= constant_expression]
[,parameterName2 [:= constant_expression],…];
```

Components

Component	Description
`typeName`	The datatype of the declared parameter.
`parameterName`	The name of the declared parameter.
`constant_expression`	An expression that evaluates to a constant.

Usage

Parameters are defined using the qualifier `parameter`. Optionally, you can specify a default value. The default value is used only if no value is provided for the parameter at server start-up.

Parameters can use only basic datatypes, and are declared in the global DECLARE block of a project or a module. Parameters cannot be declared with complex datatypes. Since parameters are constant, their value cannot be changed in the model.

Parameters at Project Setup

You can define parameters inside the global declare block for both a project and a module. Project-level parameters can be bound on server start-up, while module-level parameters are bound when the module is loaded.

You can assign values to parameters at server start-up time on the command line used to start the server or through the project configuration file. Provide values for any project parameters that do not have a default value. Parameters can only be bound to a new value when a module or project is loaded.

Specify a default value in the parameter declaration. The default value is used for the parameter if it is not bound to a new value when the project or module is loaded. If a parameter does not have a default value, it is bound when the module or project is loaded, or an error occurs.

When a parameter is initialized with an expression, that expression is evaluated only at compile time. The parameter is then assigned the result as its default value.

When supplying values at runtime for a parameter declared as an interval datatype, interval values are specified with the unit notation in CCL and with a bare microsecond value in the project configuration file. See the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for more information on project configurations and parameters in the project configuration file.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide

5.5.3 Variables

Variables represent a specific piece of information that may change throughout project execution. Variables are declared using the CCLScript syntax.

Syntax

```
typeName {variableName[:=any_expression] [, ...]}
```

Usage

Variables may be declared within any declare block, CCLScript UDFs, or Flex procedures. Multiple variables may be declared on a single line.

The declaration of a variable can also include an optional initial value, which must be a constant expression. Variables without an initial value initialize to NULL.

Variables can be of complex types. However, complex variables can only be used in local declare blocks and declare blocks within a Flex stream.

Variables declared in a local declare block may subsequently be used in SELECT clauses, but cause compiler errors when used in WHERE clauses.

Example

This example defines a variable, then uses the variable in both a regular stream and a Flex stream.

```
declare
 INTEGER ThresholdValue := 1000;
end;
//
// Create Schemas
Create Schema TradeSchema(
    Ts bigdatetime,
    Symbol STRING,
    Price MONEY(4),
    Volume INTEGER
);
Create Schema ControlSchema (
    Msg STRING,
    Value   INTEGER
); //
// Input  Trade Window
//
CREATE  INPUT  WINDOW TradeWindow
  SCHEMA TradeSchema
  PRIMARY KEY (Ts);
//
// Input Stream for Control Messages
```

```
//
CREATE INPUT STREAM ControlMsg SCHEMA ControlSchema ;
//
// Output window, only has rows that were greater than the thresholdvalue
// was when the row was received
CREATE  Output  WINDOW OutTradeWindow
    SCHEMA (Ts bigdatetime, Symbol STRING, Price MONEY(4), Volume INTEGER)
    PRIMARY KEY (Ts)
as
select *
    from TradeWindow
    where TradeWindow.Volume > ThresholdValue;
//
//Flex Stream to process the control message
CREATE FLEX FlexControlStream
  IN ControlMsg
  OUT OUTPUT WINDOW SimpleOutput
  SCHEMA ( a integer, b string, c integer)
     PRIMARY KEY ( a)
BEGIN
    ON ControlMsg
    {
        // change the value of ThresholdValue
        if ( ControlMsg.Msg = 'set')  {ThresholdValue:=ControlMsg.Value;}
        // The following is being populate so you can see that the
ThresholdValue is being set
        output [a=ControlMsg.Value; b=ControlMsg.Msg; c=ThresholdValue; |];
    }
    ;
END;
```

5.5.4 Declaring Project Variables, Parameters, Datatypes, and Functions

Declare variables, parameters, typedefs, and functions in both global and local DECLARE blocks.

Procedure

1. Declare a parameter using the SAP HANA Streaming Development perspective visual editor:
   a. In the Outline view, right-click Statements or one of its child folders (those directly below it) and select **Modify** > **Edit Global Declaration(s)**.
   b. In the Edit Expression Value pop-up window, enter the new parameter. To see a list of datatypes, press **Ctrl+Space**.
   The new parameter is visible under Statements > Globals > DeclareGlobalBlock.

   > Note
   >
   > You can double-click on either DeclareGlobalBlock or the current parameter listed to launch the Edit Expression Value pop-up window.

2. To declare parameters, variables, or user-defined CCLScript functions using CCL:
   a. Create a global declare block for your project by using the DECLARE statement in your main project file.

b. Add parameters, variables, or user-defined CCLScript functions to the global declare block.

   Elements defined in this declare block are available to any elements in the project that are not inside a module.
c. Create local declare blocks by using the DECLARE statement within derived streams, windows, or both.
d. Add variables, parameters, or user-defined CCLScript functions to the local declare block.

   These elements are available only from within the stream, window, or Flex operator in which the block is defined.

5.6 Flex Operators

Flex operators provide extensibility to CCL, allowing custom event handlers, written in CCLScript, to produce derived streams or windows.

A Flex operator produces derived streams, windows, or keyed streams in the same way that a CREATE statement produces these elements. However, a CREATE statement uses a CCL query to derive a new window from the inputs, whereas a Flex operator uses a CCLScript script.

Flex operators make CCL extensible, allowing you to implement event processing logic that would be difficult to implement in a declarative SELECT statement. CCLScript gives you process control and provides data structures that can retain state from one event to the next.

All of the features of CCLScript are available for use in a Flex operator, including:

- Data structures
  - Variables
  - EventCache (windows)
  - Dictionaries
  - Vectors
- Control structures
  - While
  - If
  - For

A Flex operator can take any number of inputs, and they can be any mix of streams, keyed streams, or windows. You can write a CCLScript event handler for each input. When an event arrives on that input, the associated CCLScript script or method is invoked.

You do not need a method for every input. Some inputs may merely provide data for use in methods associated with other inputs; for inputs without an associated method, incoming events do not trigger an action, but are available to other methods in the same Flex operator.

Use multiple output statements to process an event; the outputs are collected as a transaction block. Similarly, if a Flex operator receives a transaction block, the entire transaction block is processed and all output is collected into another transaction block. This means that downstream streams, and the record data stored within the stream, are not changed until the entire event (single event or transaction block) is processed.

In this section:

5.6.1 Creating a Flex Operator in the Visual Editor

Create a Flex operator to add an event handler written in CCLScript to the project.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor palette, in **Streams and Windows** , select **Flex** ().
3. Click anywhere in the diagram to place the Flex operator.
4. To set the name of the Flex operator, choose one of the following:
    - Click and press **F2** to edit the operator name.
    - In verbose mode, click the edit icon next to the name.
5. Connect the Flex shape to the appropriate input streams or windows.

    > **Note**
    >
    > When you connect a stream or window to a Flex operator, the source is added as an input to the Flex shape by default, and an On Input method is created from the source stream or window.

6. Click **Add Columns** to define the schema of the events produced by the Flex operator, or set the schema to a named schema in the Properties View.
7. For each input to the Flex operator, the visual editor automatically adds a null input method. To add input methods without first connecting the Flex shape to an input, use the **Add On Input Method** in the shape toolbar.
    Each method is a CCLScript script that is invoked when an event arrives on the associated input. In other words, these are event handlers.
    a. To edit the CCLScript script for each method, select the Flex shape and press **F6** to switch to the CCL editor.
    The CCL editor opens with the cursor at the CREATE FLEX statement.
    b. Edit the CCLScript script.
    c. Press **F6** to switch back to the visual editor.
8. (Optional) Add an aging policy.
9. (Optional) Click **Set Output Keep Policy** and set keep policy options.

5.7 Modularity

A module in SAP HANA smart data streaming offers reusability; it can be loaded and used multiple times in a single project or in many projects.

Modularity organizes project elements into self-contained, reusable components, or modules, which have well-defined inputs and outputs, and allow you to encapsulate data processing procedures that are commonly repeated.

Modules, along with other objects such as import files and the main project, have their own scope, which defines the visibility range of variables or definitions. Any variables, objects, or definitions declared in a scope are available within that scope only; they are unavailable to the containing scope—called the parent scope—or to any other outer scope. The parent scope can be a module or the main project. For example, if module A loads module B and the main project loads module A, then module A's scope is the parent scope to module B. Module A's parent scope is the main project.

Modules have explicitly declared inputs and outputs. Inputs to the module are associated with streams or windows in the parent scope, and outputs of the module are exposed to the parent scope using identifiers. When a module is reused, any streams, variables, parameters, or other objects within the module replicate, so that each version of the module exists separately from the other versions.

You can load modules within other modules, so that module A can load module B, which can load module C, and so on. Module dependency loops are not valid. For example, if module A loads module B, which loads A, the CCL compiler generates an error indicating a dependency loop between modules A and B.

When you load a module, you can connect or bind its input streams or windows to streams in the project. The output of a module can be exposed to its parent scope and referenced in that scope using the aliases provided in the LOAD MODULE statement.

Parameters inside the module are bound to parameters in the parent scope or to constant expressions. Stores within the module are bound to stores in the parent scope. Binding a store within a module to a store outside the module means that any windows using the module store instead use the bound store.

> **Note**
>
> All module-related compilation errors are fatal.

In this section:

[Configuring a Loaded Module [page 151]](#)
: Add or remove input and output bindings and parameter values (if any) for a specific module instance.

[Configuring a Module Repository [page 152]](#)
: Create a folder in which to store modules, and configure the smart data streaming plugin for SAP HANA studio to use it.

Related Information

Sample Project with Modules [page 326]

5.7.1 Using a Module Within a Streaming Project

Create an instance of a defined module within a streaming project, and allow the inputs and outputs of the module to be bound to streams or windows in the project.

Context

Existing modules, either created within the project or imported, can be used anywhere in a project. When you use (load) a module in a project, you attach the module inputs and outputs to streams or windows in the project by configuring bindings, and set any parameters used in the module.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the **Module** drawer of the visual editor palette, locate and select the module to add to the project.
   The palette lists any modules defined in the current project, either in the main CCL file or in any imported CCL files. If studio does not find any CREATE MODULE statements, the palette drawer is empty.
3. Click anywhere in the diagram to place the loaded module.

5.7.2 Creating a Module in the Visual Editor

Add a new module to an existing project in the visual editor.

Context

Create modules directly in a project when you do not plan to reuse them widely across other projects.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor palette, in Shared Components, select **Module** ().
3. Select a location in the diagram and click to add the shape.

Next Steps

Open the module to edit it by clicking the **Open Module Diagram** in the toolbar of the module shape. This opens a new diagram where you can add input streams or windows, simple queries, and derived streams or windows. When finished, return to the diagram that has the CREATE MODULE shape, and configure the inputs and outputs, selecting from the elements defined in the module.

5.7.3 Editing a Module in the Visual Editor

Edit basic module properties and module input, output, and import functions.

Prerequisites

Create a module.

Context

Specific module inputs and outputs are determined by project developers. Imported modules have restrictions on editing, but you can modify module inputs and outputs.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor, select the module to edit.
3. To edit the module name to be unique across all object names in the scope for this module, choose one of the following:
    - Click the module name.
    - In verbose mode, click **Edit** .
    - Select the module, and in the Properties view modify the **name** value.

    By default, the properties view is in the lower left of the SAP HANA Streaming Development perspective.
4. Click **Add Module Exported Reference(s)** ( ).
5. In the Module Exported Reference(s) dialog, select the reference(s) to add or remove, then click **OK**.
6. Click **Add Module Inputs** ().
7. In the Module Inputs dialog, select the inputs to add or remove, then click **OK**.
8. Select **Add Module Outputs** ().
9. In the Module Outputs dialog, select the outputs to add or remove, then click **OK**.
10. To access and edit the contents of the CREATE MODULE statement, select **Open Module Diagram** ().
11. Edit the module in the diagram that opens.
12. Add comments in the properties view.

5.7.4 Creating a Reusable Module File

Create a new, separate module file that can be imported into a project.

Context

You can create modules within a project, or in separate files that you can then import into a project. Create separate module files if you are likely to reuse a particular module often, in different projects. Module files are CCL files hold an individual CREATE MODULE statement.

Procedure

1. Choose **File** > **New** > **CCL Module File** .
2. Enter a unique file name.

    This becomes the module name, and is unique across all object names in the scope for this module.

3. (Optional) Specify a different folder.
   By default, the module is created in the workspace for the current project.
4. Modify the module as required and save.
   To edit the CCL, see *CREATE MODULE Statement* in the *SAP HANA Smart Data Streaming: CCL Reference*.

Related Information

SAP HANA Smart Data Streaming: CCL Reference

5.7.5 Importing Definitions from Another CCL File

Import a module file to use the module in your project.

Context

You can do this either in the CCL editor using the IMPORT statement, or by using the visual editor, as described here.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. Open the visual editor by clicking **Switch to Visual**, or pressing **F4**.
3. If Outline view is not visible, select **Window > Show View > Outline**, or press **Alt+Shift+O**.
4. In the Outline view, expand the **Statements** list.
5. Right-click the **Imports** statement and select **Create Child > Import**.
6. Select the file or files to import and click **OK**.
7. Expand the imported file until you see the imported module.
8. Click and drag the module anywhere in the diagram.

5.7.6 Configuring a Loaded Module

Add or remove input and output bindings and parameter values (if any) for a specific module instance.

Context

Active modules are created when existing module definitions are used to create new module instances.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the diagram, select the module instance to edit.
3. To edit the name of the module instance, either:
   - Click the load module instance name.
   - In verbose mode, click **Edit** .
4. Set the input bindings by adding connectors:
   a. Expand the Input Bindings compartment to that you can see the list of inputs.
   b. Add connectors to the shape in the order of the list of inputs. To see the schema for an input or how a particular input is used in the module, you can look "inside" the module by clicking the **Open Module Diagram** on the shape toolbar. This opens the model in a separate editor so that you can see the structure of the module.
5. Output bindings are set automatically, and the outputs appear on the diagram attached to the module instance. Rename the outputs as desired.

   > **Note**
   >
   > For input bindings, the schema on both sides of the binding needs to be compatible.

6. (Optional) Modify input or output bindings by selecting an individual binding in the loaded module, and changing any of the following options in the Properties window:

Property	Value
inputStreamOrWindow	Select the available input stream or window components from the list.
streamOrWindowInModule	Select the available stream or window to bind with exist-ing stream or window inputs.
comment (Output only)	Add a comment or description of the output stream.
name (Output only)	Add a name to the output stream.

7. If the module uses any parameters, parameter bindings are listed in the module instance shape on the diagram. Set parameter values in the Properties View:

parameterInModule	the parameter name.
parameterValue	the value to set this parameter to, for this instance of the module.

8. (Optional) Click **Add Store Binding** ( ). If you omit a store binding, the default memory store is used. You can also specify a store for windows in the module.
9. Edit the store binding by selecting and modifying the available fields in the Properties window:

storeInModule	the classification of the string, by default NULL.
storeValue	value phrase that defines the parameter binding.

10. To access input or output windows used inside a loaded module, select **Open Module Diagram** ( ).

5.7.7 Configuring a Module Repository

Create a folder in which to store modules, and configure the smart data streaming plugin for SAP HANA studio to use it.

Context

Modules are reusable blocks of CCL containing one or more CREATE MODULE statements. A module repository is a directory that contains these files. Once this directory has been created and configured in studio, modules can be stored in it and loaded into projects using the SAP HANA Streaming Development perspective palette.

Procedure

1. Create a new folder, or select an existing folder, to serve as the module repository.
2. In SAP HANA Streaming Development perspective, click **Windows** > **Preferences** > **SAP HANA smart data streaming**.
3. Enter the full path to the folder you want to use as the module repository in the **Module Repository Directory** field.
4. Click **Apply**.
5. Click **OK**.

5.8 Error Streams

Error streams gather errors and the records that caused them.

Description

Error streams provide ways to capture error information along with the data that caused an error. This can assist in debugging errors during development, as well as provide real-time monitoring of projects in a production environment.

You can specify more than one error stream in a single project.

An error stream is identical to other user-defined streams, except it:

- Receives records from its source stream or window only when there is an error on that stream or window. The record it receives is the input to the source stream or window that caused the error.
- Has a predefined schema that cannot be altered by the user.

The following table describes the schema for an error stream:

Column	Datatype	Description
`errorCode`	integer	The numeric code for the error that was reported.
`errorRecord`	binary	The record that caused the error.
`errorMessage`	string	Plain text message describing the error.
`errorStreamName`	string	The name of the stream on which this error was reported.
`sourceStreamName`	string	The name of the stream that sent the record that caused the error.
`errorTime`	bigdatetime	The time the error occurred; a microsecond granularity timestamp.

Error Codes and Corresponding Values

- NO_ERR - 0
- GENERIC_ERROR - 1
- FP_EXCEPTION - 2
- BADARGS - 3
- DIVIDE_BY_ZERO - 4
- OVERFLOW_ERR - 5
- UNDERFLOW_ERR - 6
- SYNTAX_ERR - 7

Limitations

The syntax of the error stream provides a mechanism for trapping runtime errors, and has these limitations:

- Only errors that occur during record computation are captured in error streams. Errors in computations that occur at server start-up, such as evaluation of expressions used to initialize variables and parameters, are not propagated to error streams. Other errors, such as connection errors and noncomputational errors, are not captured in error streams.
- Errors that occur during computations that happen without a triggering record will propagate an error record where the errorRecord field contains an empty record. Examples include the `ON START TRANS` and `ON END TRANS` blocks of the Flex block.
- For `recordDataToRecord`, the stream name must be a string literal constant. This allows a record type of the return value to determined during compilation.
- The triggering record must be retrieved using provided schemas. No native nested record support is provided to refer to the record directly.
- The triggering record reported is the immediate input for the stream in which the error happened. This may be a user-defined stream or an intermediate stream generated by the compiler. When using `recordDataToString` and `recordDataToRecord`, the first argument must match the intermediate stream if one has been generated.
- The subscription utility does not automatically decrypt (that is, convert from binary to ASCII) the error record.
- Output adapters do not automatically decrypt (that is, convert from binary to ASCII) the error record.
- Arithmetic and conversion errors occurring in external functions (such as C and Java) are not handled. These errors are the user's responsibility.
- Error streams are not guaranteed to work within the debugger framework.

In this section:

5.8.1 Monitoring Streams for Errors

Use error streams to monitor other streams for errors and the events that cause them. You can create error streams in the studio visual editor, or directly in CCL.

Process

1. Identify the project and the specific streams to monitor.
2. Decide whether to use multiple error streams, then determine the visibility for each error stream.
3. Create the error streams in that project. See Creating an Error Stream in the Visual Editor [page 156].
4. Display some or all of the information from the error streams in the error record; that is, information aggregated or derived from the error records. See Displaying Error Stream Data [page 157].

5.8.2 Examples: Error Streams in CCL

View CCL examples of error streams in use.

In a project that has one input stream and two derived streams, create a locally visible error stream to monitor all three streams using:

```
CREATE ERROR STREAM AllErrors ON InputStream, DerivedStream1, DerivedStream2;
```

To keep a count of the errors according to the error code reported, add:

```
CREATE OUTPUT WINDOW errorHandlerAgg SCHEMA (errorNum integer, cnt long)
PRIMARY KEY DEDUCED
AS
SELECT e.errorCode AS errorNum, COUNT(*) AS cnt
FROM AllErrors e
GROUP BY e.errorCode;
```

In a project that has three derived streams, create an externally visible error stream to monitor only the third derived stream (which calculates a volume weighted average price) using:

```
CREATE OUTPUT ERROR STREAM vwapErrors ON DerivedStream3;
```

To convert the format of the triggering record from binary to string, add:

```
CREATE OUTPUT STREAM vwapMessages SCHEMA (errorNum integer, streamName string,
errorRecord string) AS
SELECT  e.errorCode AS errorNum,
        e.errorStreamName AS streamName,
        recordDataToString(e.sourceStreamName, e.errorRecord) AS errorRecord
FROM vwapErrors e;
```

5.8.3 Creating an Error Stream in the Visual Editor

Add error streams using the studio visual editor.

Context

Whether you are debugging a project in development or monitoring a project in production mode, error streams let you see errors and the records that cause them in other streams in real time.

> **Note**
>
> An error stream cannot monitor other error streams.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor, open the project.
3. Click the error stream shape in the palette, then click an empty area in the diagram.
4. Click the **+** (plus) sign.
   You see a list of streams in the project that can be monitored.
5. Specify the streams you want to monitor: click **Select All** or click each stream to monitor, then click **OK**.
   The streams you specified are connected to the error stream by red lines indicating that they are sending error information.

5.8.4 Modifying an Error Stream

When you are debugging a project in development or monitoring a project in production mode, you can change the specific streams that an error stream is monitoring.

Context

> **Note**
>
> An error stream cannot monitor other error streams.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor, open the project.
3. Locate the error stream shape in the work area and review the list of input streams.
4. Click the **+** (plus) sign, then click each stream to monitor, and click **OK**. Or, use the Connector in the palette to connect an input stream to the error stream.
   A red line connects each stream to the error stream and the new stream names appear on the Inputs list.
5. (Optional) To remove input streams from the error stream, click the X in a red circle, then select each stream to remove. Click **OK**.
   The red lines connecting the streams to the error stream and the stream names on the Inputs list are removed.

5.8.5 Displaying Error Stream Data

By default, error streams created in studio are output. However, if you configure error streams as local, or create them directly in CCL (where the default is local), you can change them back to output to make them visible to external subscribers. This makes real-time monitoring of the error streams possible.

Context

In production mode, project monitoring may be performed externally.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor, open the project.
3. To enable real-time monitoring of errors encountered by the project, click the **Type** icon in the Error Stream to toggle it from OUTPUT to LOCAL.
4. To enable ad hoc SQL queries, add a window (for example, ErrorState) to the project, downstream from the error stream.
   The ErrorState window preserves the state of the error stream so it can be queried using the `streamingquery` utility. See *streamingquery* in the *SAP HANA Smart Data Streaming: Utilities Guide* for an example of using `streamingquery` to check the ErrorState.

Related Information

SAP HANA Smart Data Streaming: Utilities Guide

5.9 Data Retention and Recovery with Stores

Every window is assigned to a store, which holds the retained records. By default, all windows are assigned to a memory store. You can create log stores to add data recoverability and to optimize performance, then assign windows to specific stores.

Memory Stores

A memory store holds all data in memory. Memory stores retain the state of queries for a project from the most recent server start-up for as long as the project is running. Because query state is retained in memory rather than on disk, access to a memory store is faster than to a log store.

Use the CREATE MEMORY STORE statement to create memory stores. If no default store is defined, new windows are automatically assigned to a memory store.

Log Stores

The log store holds all data in memory, but also logs all data to the disk, guaranteeing data state recovery in the event of a failure. Use a log store to recover the state of a window after a restart.

Use the CREATE LOG STORE statement to create a log store. You can also set a log store as a default store using the CREATE DEFAULT LOG STORE statement, which overrides the default memory store.

Log store dependency loops can cause compilation errors when using log stores. You might create log store loops when you use multiple log stores in a project, and assign windows to these stores. For the best results, assign log stores to source windows only, or to assign all windows in a stream path to the same store. If you use `logstore1` for `n` of those windows, then use `logstore2` for a different window, do not reuse `logstore1` further down the chain. For example, if Window Y assigned to Logstore B gets its data from Window X assigned to Logstore A, no window that (directly or indirectly) gets its data from Window Y should be assigned to Logstore A.

In this section:

[Using Log Stores for Data Recovery [page 159]]
A log store provides data recovery inside a window if a server fails or is shut down unexpectedly.

[Creating a Log Store [page 160]]
If failover is enabled, configure a log store to capture the data that flows through a project.

[Creating a Memory Store [page 171]]
Create a memory store to retain the state of continuous queries in memory, from the most recent server startup.

5.9.1 Using Log Stores for Data Recovery

A log store provides data recovery inside a window if a server fails or is shut down unexpectedly.

Properly specified log stores recover window elements on failure, and make sure data gets restored correctly if the server fails and restarts. You can use log stores with windows that have no retention policy; you cannot use log stores with stateless elements.

When using log stores, consider:

- Log stores only store window contents.
- Log stores do not directly store intermediate state, such as variables.
- Local Flex stream variables and data structures are not directly stored. However, they may be regenerated from source data if the source data is in persistent storage.
- Log stores do not preserve opcode information. (During periodic log store compaction and checkpointing, only the current window state is preserved. Records are then restored as inserts.)
- Row arrival order is not preserved. In any stream, multiple operations may be collapsed into a single record during log store compaction, changing arrival order. Inter-stream arrival order is not maintained.
- You can define one or more log stores in a project. When using multiple stores, make sure you prevent the occurrence of log store loops, which cause compilation errors. A log store loop is created when, for example, `Window1` in `Logstore1` feeds `Window2` in `Logstore2`, which feeds `Window3` in `Logstore1`.
- The contents of memory store windows that receive data directly from a log store window are recomputed once the log store window is restored from disk.
- The contents of memory store windows that receive data from a log store window via other memory store windows are also recomputed, once the input window's contents have been recomputed.
- In the case of partitioning, if the input of the partition target is a stream, which is a stateless element, then operations such as filter, compute, aggregate, and join are not supported.
- If the input of a partitioned target is on a memory store and the target is on a log store, this is supported only if the memory store (input element) can recover its data from an element that is on a log store.

> **Note**
>
> If a memory store window receives data from a log store window via a stateless element such as a stream, its contents are not restored during server recovery.

When you shut down the server normally, it performs a quiesce and checkpoint before it shuts down. It is therefore able to store all data currently in the project, as the data has been fully processed and is in a stable state. When an abnormal system shutdown occurs between checkpoints, there is no way of knowing the state of the system or where the uncheckpointed data was. Therefore, the uncheckpointed data on the input windows attached to log stores is replayed by streaming events down the project as though they were going through the input windows for the first time. The uncheckpointed data is replayed in an attempt to attain a state as close as possible to the state of smart data streaming before the abnormal shutdown.

Log stores are periodically compacted, at which point all data accumulated in the store is checkpointed and multiple operations on the same key are collapsed. After a checkpoint, the store continues appending incoming data rows to the end of the store until the next checkpoint.

> **Note**
>
> The recovery of data written to the store, but not yet checkpointed, is available only for input windows. When you assign a window to a log store, you should also assign all of its input windows to a log store. Otherwise, data written to the window after the last checkpoint is not restored.

Unlike memory stores, log stores do not extend automatically. Use the CCL `maxfilesize` property to specify the log store size. The size of a log store is extremely important. Log stores that are too small can cause processing to stop due to overflow, and can also cause significant performance degradation due to frequent cleaning cycles. A log store that is too large can hinder performance due to larger disk and memory requirements.

Parent topic: Data Retention and Recovery with Stores [page 158]

Related Information

Creating a Log Store [page 160]
Creating a Memory Store [page 171]

5.9.2 Creating a Log Store

If failover is enabled, configure a log store to capture the data that flows through a project.

Prerequisites

- Determine the size of your log store.
- Review Log Store Guidelines [page 162].

Context

> **Note**
>
> Log stores do not store SAP HANA smart data streaming event logs (cluster logs, server logs, or project logs).

Create one log store per project. The preferred location for log store files is the base directory where project files are stored.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the CCL editor, create a log store using the CREATE LOG STORE statement:

```
CREATE [DEFAULT] LOG STORE storename
PROPERTIES
filename='filepath'
     [sync={ true | false},]
     [sweepamount=size,]
     [reservepct=size,]
     [ckcount=size,]
     [maxfilesize=filesize];
```

3. For the filename property, enter either a relative (preferred) or absolute file path for the location of the log store:

Option	Description
Relative path (recommended)	A relative path is relative to the smart data streaming sandbox base directory. Using a relative path means that your log store automatically points to the sandbox base directory. Relative paths do not point to the directory stack; this means that the path does not start with a drive letter or slash (/).

Absolute path (not recommended)	Absolute paths are changed into relative paths, relative to the sandbox base directory, and interpreted as literally as possible. For Windows systems, an absolute path begins with the drive letter; on UNIX systems, the absolute path begins with a slash (/). For Windows, drive letters are turned into a folder and preserved in the sandbox base directory. For example, a sandbox base directory of work1/streaming/sandbox with log stores with filename parameters of D:/streaming/logstore and data/logstore running in project importantp1 in workspace worksp1 instance number 0 becomes:
	◦ For D:/streaming/logstore:
	  ◦ work1/streaming/sandbox/store/d/streaming/logstore/worksp1.importantp1.0
	◦ For data/logstore:
	  ◦ work1/streaming/sandbox/store/data/logstore/worksp1.importantp1.0

The relative path location must be a shared disk available to all cluster nodes. The log store path is specified in the filename property within the log store definition. Both a relative path and absolute path automatically place log stores under <SAP HANA-install-directory>/data_streaming/<SID>/STREAMING_SHARED/store/<filename-property>/<workspace-name>.<project-name>.<instance-number>.

4. Enter appropriate values for the remaining properties in the CREATE LOG STORE statement.
5. Click **Compile** (F7).
6. Click **Run Project**.

In this section:

Log Store Guidelines [page 162]
There are several considerations to review before adding a log store to your project.

Sizing a Log Store [page 163]
Calculate the size of the log store your project requires. Correctly sizing your log store is important because stores that are too small or large can lead to performance issues.

Log Store Sizing Reference [page 167]

Set sizing parameters for a log store using a CREATE LOG STORE statement.

Log Store Optimization Techniques [page 170]

Specify persistence to optimize data models for maximum performance.

Task overview: Data Retention and Recovery with Stores [page 158]

Related Information

Using Log Stores for Data Recovery [page 159]
Creating a Memory Store [page 171]

5.9.2.1 Log Store Guidelines

There are several considerations to review before adding a log store to your project.

General Guidelines

- Place log stores on a shared drive available to all the machines in the cluster.
- Keep windows that change at substantially different rates in different log stores. If a log store contains a large but nearly-static stream and a small but rapidly changing stream, each cleaning cycle must process large amounts of data from the static stream. Keeping streams separate optimizes cleaning cycles.
- Put any window into a log store that:
  - Is fed by stateless elements (streams).
  - Is fed by more than one upstream source in the project data flow. This is necessary for recovery because the arrival order of rows is not preserved.
  - Cannot produce the same result before and after a disruptive event such as a server crash, based on data replayed during the recovery process.
- Log stores use window names internally for identification. Start a new file for a log store when renaming a window to which it is attached.
- Variables and CCLScript data structures (dictionaries, vectors, and event caches) do not persist in log stores and cannot be recovered after a failure. Use these structures with log stores only when:
  - You can provide logic to reconstruct the structures on restart.
  - Processing is not affected if the structures are missing after a restart.

Guidelines for Guaranteed Delivery

All the general guidelines above apply to log stores for windows with guaranteed delivery. In addition:

- Ensure that the log store for every guaranteed delivery stream or window is large enough to accommodate the required events. If the log store runs out of room, the project server shuts down. Because copies of events are kept in the same log store the stream or window is assigned to, the size of the log store for a guaranteed delivery stream or window must be significantly larger than that for a similar stream or window without guaranteed delivery.
- Put any window on which GD is enabled, and all input windows that feed GD windows, into a log store. You can put windows located between the input and GD windows in a memory store if, upon restart, they can be reconstructed to exactly the same state they were in before the server went down. If an intermediate window cannot be reconstructed to its previous state, put it in a log store.
  - If consistent recovery is not enabled, put the GD windows and all their feeder windows into the same log store. Note, however, that placing many windows in the same log store adversely affects performance.
  - If consistent recovery is enabled, you can employ as many log stores for your GD and feeder windows as necessary.

Parent topic: Creating a Log Store [page 160]

Related Information

Sizing a Log Store [page 163]
Log Store Sizing Reference [page 167]
Log Store Optimization Techniques [page 170]

5.9.2.2 Sizing a Log Store

Calculate the size of the log store your project requires. Correctly sizing your log store is important because stores that are too small or large can lead to performance issues.

Prerequisites

Review Log Store Guidelines [page 162].

Context

Start this procedure by calculating your project's internal record size. An internal record represents a row in a window. Each row contains a fixed-size header plus a variable-size payload containing the column offsets, column data, and any optional fields. Use this formula for the calculation in the first step of this procedure:

$$\text{HeaderSize}(56) + \text{Offsets}(4 * \text{M}) + \sum_{1}^{M} \text{PS}$$

In the formula:

- M represents the number of columns
- PS represents the primitive datatype size for each of the M columns

Primitive datatypes are the building blocks that make up more complex structures such as records, dictionaries, vectors, and event caches. The following table provides the size of primative datatypes:

Datatype	Size in Bytes	Notes
Boolean	1	
Decimal	18	
Integer	4	
Long	8	
String	1 + number of characters in the string	Estimate an average length
Float	8	
Money(n)	8	
SecondDate	8	
Time	8	
MsDate	8	
BigDateTime	8	
Binary	4 + number of bytes in the binary value	Estimate an average length

Note

Guaranteed delivery (GD) logs hold events stored for delivery. If no GD logs are stored in the log store, you may skip the first three steps of this procedure. Instead, compute the dataSize using the Playback feature in studio or the `streamingplayback` utility to record and play back real data to get a better idea of the amount of data you need to store. (See Playback View [page 117] for details on Playback or the *SAP HANA Smart Data Streaming: Utilities Guide* for details on `streamingplayback`.) The log store reports "`liveSize`" in the project log file (`streamingserver~<workspace-name>.<project-name>.<project-instance-number>~_<machine-hostname>.<cluster-node-rpc-port>.<log-serial-number>.trc`) when the project exits (with log level three or higher) or after every compaction (with log level six or higher). Use the "`liveSize`" value for the dataSize referenced in step 2 of this procedure and beyond.

Procedure

1. (Optional if no GD logs are stored in the log store) For each window, calculate the size of an internal record. If the window supports GD, compute the size for the GD logs separately.

For purposes of illustration, use this schema:

```
CREATE SCHEMA TradesSchema AS (
       TradeId    LONG,
       Symbol     STRING,
       Price      MONEY(4),
       Volume     INTEGER,
       TradeDate  BIGDATETIME
);
```

a. Using the primitive sizes from the Primitive Datatype Sizes table above, compute the column values—the total size in bytes for the datatypes in the schema. For the sample schema, assuming an average STRING length of 4, the calculation is:

```
8 + (4 + 1) + 8 + 4 + 8 = 33 bytes
```

b. Add the size of the offsets to the size of the column values. The offsets are calculated as (4 * M) where M is the number of columns. Plugging in the sample schema's five columns, we get:

```
(4 * 5) + 33 = 53 bytes
```

c. Add the size of the row header, which is always 56 bytes:

```
56 + 53 = 113 bytes
```

d. Round up to the nearest number divisible by:
   - 8 if smart data streaming is running on a 64-bit architecture
   - 4 if smart data streaming is running on a 32-bit architecture

   For a 64-bit installation, use this formula, where URS is the unrounded record size value you computed in step 1:

```
URS + (8 - (URS modulo 8))
```

   (For a 32-bit installation, substitute a 4 for each 8 in the formula.) Continue with the example, where smart data streaming is running on a 64-bit machine:

```
113 + (8 - (1)) = 120 bytes
```

e. Label your result recordSize and make a note of it.

2. Estimate the maximum amount of data, in bytes, that you expect to collect in the log store. To do this, determine the maximum number of records each window assigned to the log store will contain. If the window supports guaranteed delivery, treat the GD logs as a separate window. For the record count, use the maximum number of uncommitted rows you expect the GD logs to contain for this window. Add 1000 to this value, since GD logs are purged only when there are at least 1000 fully committed events.

   a. For each window, determine the data size by multiplying the expected record count by the recordSize you computed in step 1.
   b. Sum the data size for all the windows and GD logs to get the total size of the data that will be stored in the log store. Label this value dataSize.
   c. Sum the record counts for each window and GD log assigned to this log store and label that value recordCount.

3. (Optional if no GD logs are stored in the log store) To calculate the basic indexing overhead, multiply the recordCount from the previous step by 96 bytes. Add the result to the dataSize value.

4. Choose the value of the `reservePct` parameter. Calculate the required store size, in bytes, including the reserve, as follows, where dataSize is the value you computed in the previous step:

   storeBytes = dataSize * 100 / (100 - `reservePct`)

   Round storeBytes up to the next megabyte.
5. Ensure the reserve cannot be overrun by the uncheckpointed data.

   Estimate the maximum amount of uncheckpointed data that is produced when the input queues of all the streams, except source streams, are full. Count the records in the queues that are located early in the sequence together with any records they produce as they are processed through the project. Include the number of output records produced by the stream for each of its input records.

   This example shows the stream queue depth set to the default of 1024, for a log that contains four streams ordered like this:

   ```
   source --> derived1 --> derived2 --> derived3
   ```

   a. Determine the number of records produced by each stream as it consumes the contents of its queue:
      - 1024 records may end up in `derived1`'s input queue. Assuming the queue produces one output record for one input record, it produces 1024 records.
      - 2048 records may end up in `derived2`'s input queue (1024 that are already collected on its own queue, and 1024 more are `derived1`). Assuming that `derived2` is a join and generates an average of 2 output records for each input record, it produces 4096 records ([1024 + 1024] * 2).
      - 5120 records may end up in `derived3` (1024 from its own queue and 4096 from `derived2`). Assuming a pass-through ratio of 1, `derived3` produces 5120 records.

      Take all branches into account when the project's topology is not linear. The pass-through ratio may differ for data coming from the different parent streams. Add up the data from all the input paths. Each stream has only one input queue, so its depth is fixed, regardless of how many parent streams it is connected to. However, the mix of records in each queue may vary. Assume the entire queue is composed from the records that produce that highest amount of output. Some input streams may contain static data that is loaded once and never changes during normal work. You do not need to count these inputs. In the example, `derived2` is a join stream, and has static data as its second input.
   b. Calculate the space required by multiplying the total number of records by the average record size of that stream.
      For example, if the records in `derived1` average 100 bytes; `derived2`, 200 bytes; and `derived3`, 150 bytes, the calculation is:

      (1024 * 100) + (4096 * 200) + (5120 * 150) = 1,689,600

      Trace the record count through the entire project, starting from the source streams down to all the streams in the log store. Add the data sized from the streams located in the log store.
   c. Multiply the record count by 96 bytes to calculate the indexing overhead and add the result to the volume in bytes:

      (1024 + 4096 + 5120) * 96 = 983,040

      1,689,600 + 983,040 = 2,672,640

      Verify that this result is no larger than one quarter of the reserve size:

      uncheckpointedBytes < storeBytes * (`reservePct` / 4) / 100

If the result is larger than one quarter of the reserve size, increase the reserve percent and repeat the store size calculation. Uncheckpointed data is mainly a concern for smaller stores. Other than through the uncheckpointed data size, this overhead does not significantly affect the store size calculation, because the cleaning cycle removes it and compacts the data.

6. When you create the log store, place the value you arrived at for your log store size here, in the CREATE LOG STORE statement's `maxfilesize` parameter.

Task overview: Creating a Log Store [page 160]

Related Information

Log Store Guidelines [page 162]
Log Store Sizing Reference [page 167]
Log Store Optimization Techniques [page 170]
SAP HANA Smart Data Streaming: Utilities Guide

5.9.2.3 Log Store Sizing Reference

Set sizing parameters for a log store using a CREATE LOG STORE statement.

The CREATE LOG STORE parameters described here control the size and behavior of the log store.

maxfilesize Parameter

The maximum file size is the largest size, in bytes, that the log store file is allowed to reach. To calculate this value, see Sizing a Log Store [page 163] for instructions on calculating this value.

It is important to size log stores correctly because unlike memory stores, log stores do not extend automatically. A store that is too small requires more frequent cleaning cycles, which severely degrades performance. In the worst case, the log store can overflow and cause processing to stop. A store that is too large also causes performance issues due to the larger memory and disk footprint; however, these issues are not as severe as those caused by log stores that are too small.

reservePct Parameter

The reserve is intermediate or free space maintained in every log store, which is used when the store is resized and during periodic cleaning of the store. The `reservePct` value is a percentage of the size of the log store.

> **Note**
>
> If the reserve space is too small and the project runs until the store fills with data, a resize attempt may cause the store to become wedged. This means that it cannot be resized, and the data can be extracted

from it only by SAP Technical Support. It is safer to have too much reserve than too little. The default of 20 percent is adequate in most situations. Multigigabyte stores may use a reduced value as low as 10 percent. Small stores, under 30 MB, especially those with multiple streams, may require a higher reserve (up to 40 percent). If you find that 40 percent is still not enough, increase the size of the store.

SAP HANA smart data streaming automatically estimates the required reserve size and increases the reserve if it is too small. This usually affects only small stores. It is a separate operation from resizing the log store itself, which is performed by a user.

> **Note**
>
> Increasing the reserve reduces the amount of space left for data. Monitor project log messages for automatic adjustments when you start a new project. You may need to increase the store size if these messages appear.

As the store runs, more records are written into it until the free space falls below the reserve, at which point the source streams are temporarily stopped, the streams quiesced, and the checkpoint and cleaning cycle are performed. Streams do not quiesce immediately, but must first process any data collected in their input queues. Any data produced during quiescence is added to the store, meaning that the reserve must be large enough to accommodate this data and still have enough space left to perform the cleaning cycle. If this data overruns the reserve, it cannot perform the cleaning cycle and the store becomes wedged. The automatic reserve calculation does not account for data that is no checkpointed.

Log Store Size Warnings

As the amount of data in the store grows, if the free space falls below 10 percent (excluding the reserve), smart data streaming starts reporting `log store is nearing capacity` in the project log file (`streamingserver~<workspace-name>.<project-name>.<project-instance-number>~_<machine-hostname>.<cluster-node-rpc-port>.<log-serial-number>.trc`). If the data is deleted from the store in bursts (for example, if data is collected during the day, and data older than a week is discarded at the end of the day), these messages may appear intermittently even after the old data has been flushed. As the cleaning cycle rolls over the data that has been deleted, the messages disappear.

Unless your log store is very small, these warnings appear before the store runs out of space. If you see them, stop smart data streaming when convenient, and increase the store size. Otherwise, smart data streaming aborts when the free space in the project falls below the reserve size.

Recovering from a Wedged Log Store

If a log store is sized incorrectly, the entire reserve may be used up, which causes the store to become wedged. If this happens, you cannot resize the log store or preserve the content. Delete the store files and restart smart data streaming with a clean store. If you make a backup of the store files before deleting them SAP Technical Support may be able to extract content. Change the store size in the project, and it is resized on restart. You cannot decrease the store size. When you restart a project after resizing the store, it will likely produce project log messages about the free space being below the reserve until the cleaning cycle assimilates the newly added free space.

ckcount Parameter

The `ckcount` checkpointing count parameter affects the size of uncheckpointed data. This count shows the number of records that may be updated before writing the intermediate index data. Setting it to a large value amortizes the overhead over many records to make it almost constant, averaging 96 bytes per record. Setting it to a small value increases the overhead. With the count set to zero, index data is written after each transaction, and, for the single-transaction records, the overhead becomes:

96 + 32 * ceiling ($\log_2$(number_of_records_in_the_stream))

If a stream is small (for example, fewer than 1000 records), the overhead for each record is:

96 + 32 * ceiling ($\log_2$(1000)) = 96 + 32 * 10 = 416

In many cases, the record itself is smaller than its overhead of 416 bytes. Since the effect is logarithmic, large streams are not badly affected. A stream with a million records has a logarithm of 20 and incurs an overhead of 736 bytes per record. The increased overhead affects performance by writing extra data and increasing the frequency of store cleaning.

If your project includes any windows configured for guaranteed delivery (GD), consider adjusting the value of `ckcount` to improve performance and latency.

sweepamount Parameter

The `sweepamount` parameter determines how much of the log store file is "swept through" during each cleaning pass. The value is between 5 percent to 20 percent of the `maxfilesize` parameter. A good lower bound for the sweep size is half the size of the write cache on your storage array. Usually, it indicates a sweep size of 512 to 1024 megabytes. Smaller sweep sizes minimize spikes in latency at the expense of a higher average latency. High values give low average latency with higher spikes when reclaiming space.

If the value of the `sweepamount` parameter is too small, the system performs excessive cleaning; in some cases, this does not allow the log store to free enough space during cleaning.

The size of the sweep is also limited by the amount of free space left in reserve at the start of the cleaning cycle. If the reserve is set lower than the sweep amount and the sweep does not encounter much dead data, the sweep stops if the relocated live data fills up the reserve. The swept newly cleaned area becomes the new reserve for the next cycle. Unless other factors override, you should keep the sweep and the reserve sizes close to each other. `reservePct` is specified in percent, while `sweepamount` is specified in megabytes.

If your project includes any windows configured for guaranteed delivery (GD), consider adjusting the value of `sweepamount` to improve performance and latency.

Log Store Size and File Locations

Ensure the total size of all log store files does not exceed the size of the machine's available RAM. If this occurs, the machine takes longer to process the data, causing all monitoring tools to display low CPU utilization for each stream, and standard UNIX commands such as `vmstat` to display high disk usage due to system paging.

For storing data locally using log stores, you should use a high-speed storage device, for example, a redundant array of independent disks (RAID) or a storage area network (SAN), preferably with a large dynamic RAM cache. For a moderately low throughput, place backing files for log stores on single disk drives, whether SAS, SCSI, IDE, or SATA.

Parent topic: Creating a Log Store [page 160]

Related Information

Log Store Guidelines [page 162]
Sizing a Log Store [page 163]
Log Store Optimization Techniques [page 170]

5.9.2.4 Log Store Optimization Techniques

Specify persistence to optimize data models for maximum performance.

- Whenever possible, create a small log store to store static (dimension) data, and one or more larger log stores for dynamic (fact) data.
- If you are using multiple log stores for larger, rapidly changing, dynamic (fact) data, try to organize the stores on different RAID volumes.
- The correct sizing of log stores is extremely important.

Parent topic: Creating a Log Store [page 160]

Related Information

Log Store Guidelines [page 162]
Sizing a Log Store [page 163]
Log Store Sizing Reference [page 167]

5.9.3 Creating a Memory Store

Create a memory store to retain the state of continuous queries in memory, from the most recent server startup.

Prerequisites

To ensure optimal performance, consult with your system administrator on the type, number, and index values for memory stores.

Procedure

1. In the visual editor palette, in Shared Components, click **Memory Store**.
2. Select a location in the diagram and click to add the shape.
3. Connect the memory store to a window.
4. Specify a name for the store that is unique within its scope for the project or module.
5. (Optional) Click **Set Store Properties** and modify property values. The following table describes the memory store properties you can modify:

Property	Description
Index Size Hint (KB) (INDEXSIZE-HINT)	(Optional) Determines the initial number of elements in the hash table, when using a hash index. The value is in units of 1024. Setting this higher consumes more memory, but reduces the chances of spikes in latency. Default is 8KB.
Index Kind (INDEXTYPE)	The type of index mechanism for the stored elements. Default is **Tree**. Use **Tree** for binary trees. Binary trees are predictable in use of memory and consistent in speed. Use **Tree** for hash tables, as hash tables are faster, but they often consume more memory.

6. (Optional) Select **Default** to make this the default store for the project (or module).

Task overview: Data Retention and Recovery with Stores [page 158]

Related Information

Using Log Stores for Data Recovery [page 159]
Creating a Log Store [page 160]

5.10 Setting an Aging Policy

You can set an aging policy to flag records that have not been updated within a defined interval. This is useful for detecting records that may be stale.

Context

Aging policies are an advanced, optional feature for a window or other stateful elements.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor, select a window or other stateful element to which you want to add an aging policy.
3. Select **Set Aging Policy** and set values, which are described in the following table:

Value	Description
Aging Time	This is an interval value. Any record in the window that has not been updated for this much time will have the Aging Field incremented. The timer is reset when the record is updated (or the Aging Time Field changes). The period can be specified in hours, minutes, seconds, milliseconds, or microseconds.
Aging Field	The field in the record is incremented by 1 every time the aging time period elapses and no activity has occurred on the record, or until a maximum defined value is reached. The default value is 1.
(Optional) Max Aging Field Value	The maximum value that the aging field can be incremented to. If not specified, the aging field is incremented once.
(Optional) Aging Time Field	The start time of the aging process. If not specified, the internal row time is used. If specified, the field must contain a valid start time.

4. (Optional) Double-click the policy to edit its parameters.

Results

When the project runs, records accumulate until the Aging Time or Max Aging Field Value is reached. On an update to a record, the age is reset to 0.

5.11 Using Bindings to Connect CCL Projects

Create bindings between projects to let data flow between them.

You can enable a binding in a project's `.ccr` file, as long as the streams or windows you are binding have compatible schemas. See *CCL Project Basics>Bindings between CCL Projects* for more information.

Binding to a Stream on an SSL-Enabled Cluster

This example shows a binding called `BaseInputBinding` that connects a local input stream called `sin` to a remote output stream that is also called `sin`. The `<Manager>` element that specifies the cluster hostname and port in the `.ccr` file must include the https:// prefix, as shown here. If you omit the https:// prefix, the binding cannot pass data, so the input stream receives nothing.

```
<Configuration>
  <Runtime>
    <Clusters>
      <Cluster name="cluster1" type="remote">
        <Username>USER_NAME</Username>
        <Password>PASSWORD</Password>
        <Managers>
          <Manager>https://CLUSTER_MANAGER_HOSTNAME:
          CLUSTER_MANAGER_RPC_PORT</Manager>
          <!-- use https:// when SSL is enabled -->
        </Managers>
      </Cluster>
    </Clusters>

    <Bindings>
      <Binding name="sin">
        <Cluster>cluster1</Cluster>
        <Workspace>ws2</Workspace>
        <Project>prj2</Project>
        <BindingName>BaseInputBinding</BindingName>
        <RemoteStream>sin</RemoteStream>
      </Binding>
    </Bindings>
  </Runtime>
</Configuration>
```

Reconnection Intervals for Bindings

This example shows two bindings, `b1` and `b2`, on a local input stream called `MyInStream`. The `b1` binding includes a reconnection interval option specifying that if the connection between `MyInStream` and the remote output stream is lost, the project will attempt to reconnect every 10 seconds. Because the `b2` binding does not specify a reconnection interval, its reconnection attempts will occur at the default interval of five seconds:

```
<Bindings>
  <Binding name="MyInStream">
    <Cluster>c1</Cluster>
    <Workspace>w1</Workspace>
```

```
    <Project>p1</Project>
    <BindingName>b1</BindingName>
    <RemoteStream>MyInStream1</RemoteStream>
    <ReconnectInterval>10</ReconnectInterval>
  </Binding>
  <Binding name="MyInStream">
    <Cluster>c1</Cluster>
    <Workspace>w1</Workspace>
    <Project>p1</Project>
    <BindingName>b2</BindingName>
    <RemoteStream>MyInStream2</RemoteStream>
  </Binding>
</Bindings>
```

To suppress all reconnection attempts, set `<ReconnectInterval>` to 0. Use positive whole number values to set the reconnection interval in seconds.

Configuring an Input Stream or Window to Provide Output

This example shows how to configure an input stream to send data to another input stream by setting the <Output> parameter in the <Binding> element to true.

> **Note**
>
> Set the <Output> parameter to true only when you configure a binding on an input stream or window that is providing output. If you configure the binding on the stream or window that is receiving input, do not set the <Output> parameter. When you configure a binding on an output stream, you do not have to set the <Output> parameter, since output streams can only produce output.

In this example, output from the input stream `MyInStream`, in the local project, is bound to the input stream `MyInStream1` in project `p2`. The line `<Output>true</Output>` tells the binding to publish (send data out) to the remote stream. Without that line, this binding would subscribe to data from `MyInStream1` because bindings on input streams receive data by default:

```
<Binding name="MyInStream">
  <Cluster>c1</Cluster>
  <Workspace>w1</Workspace>
  <Project>p2</Project>
  <BindingName>b1</BindingName>
  <RemoteStream>MyInStream1</RemoteStream>
  <Output>true</Output>
</Binding>
```

Configuring a Window for Guaranteed Delivery

This example shows how to enable and configure guaranteed delivery (GD) on a window's output binding. The GD parameters are the same for input bindings.

Enable GD for a binding to guarantee that if the connection between the binding and the remote stream is severed (by shutting down the project that contains the local stream, for example), all transactions that are supposed to be transmitted through the binding during its downtime are processed once the connection is re-established.

Use these parameters in the <Binding> element of your `.ccr` file to set a binding to support guaranteed delivery:

- <EnableGD> – Specifies whether guaranteed delivery is enabled for this binding. Values are true and false.

  > **Note**
  >
  > When you enable GD on a binding, make sure:
  > - The binding's source data window is running in GD mode or GD mode with checkpoint.
  > - The binding's target data window is backed by a log store.

- <GDName> – Supply a unique name for the GD session (subscription) this binding establishes.
- <GDBatchSize> – The number of transactions this binding may collect in a batch before releasing the batch to the target window. The binding issues a `GD commit` to the source data window after releasing the data. This setting is ignored when the source data window is in GD mode with checkpoint and the <EnableGDCache> parameter on this binding is set to true. The default value is 10.
- <EnableGDCache> – Enable this binding to cache data. When the source data window is in GD mode with checkpoint, the binding receives checkpoint messages that indicate the last row of data that has been checkpointed by the window. If the binding is enabled for GD caching, it caches incoming transactions until it receives a checkpoint message from the source window. The checkpoint message triggers the binding to send all cached transactions up to the one indicated in the checkpoint message, to the target window. The binding issues a `GD commit` to the source data window after releasing cached data. If GD caching is disabled, checkpoint messages are ignored and the binding forwards data based on the value of <GDBatchSize>. The setting of <EnableGDCache> is ignored if the source data window is not in GD mode with checkpoint. Values are true (default) and false.

In this example, output from the local output stream `MyOutStream` is bound to `MyInStream1` in project `p1`. GD and GD caching are enabled. The GD session name is `b1_GD1` and the GD batch size is 20 transactions:

```
<Binding name="MyOutStream">
  <Cluster>c1</Cluster>
  <Workspace>w1</Workspace>
  <Project>p1</Project>
  <BindingName>b1</BindingName>
  <RemoteStream>MyInStream1</RemoteStream>
  <ReconnectInterval>5</ReconnectInterval>
  <EnableGD>true</EnableGD>
  <GDName>b1_GD1</GDName >
  <GDBatchSize>20</GDBatchSize >
  <EnableGDCache>true</EnableGDCache >
</Binding>
```

6 CCL Query Construction

Use a CCL query to produce a new derived stream or window from one or more other streams or windows. You can construct a query to filter data, combine two or more queries, join multiple datasources, use pattern matching rules, and aggregate data.

You can use queries only with derived elements, and can attach only one query to a derived element. A CCL query consists of a combination of several clauses that indicate the appropriate information for the derived element. A query is used with the AS clause to specify data for the derived element.

You can also use the generic derived stream and derived window shapes in studio to create complex queries.

Choose the element type according to your input, output, and retention requirements for data, and for preserving insert, update, and delete operations.

The following table specifies the rules for derived streams, derived windows, and derived keyed streams.

Element	Input	Output	Retains state	Preserves inserts, updates, and deletes
Derived Stream	Another stream	Stream	No	No
Derived Window	Another stream or window	Window	As defined in retention policy (default is keep all rows)	Yes **Note** To derive a window from a stream, include a GROUP BY clause in the query.
Derived Keyed Stream	Another stream or window	Stream	As defined in retention policy; otherwise no	Yes

In this section:

Use joins to combine multiple datasources into a single query.

Pattern Matching [page 191]
Use the MATCHING clause in your CCL query to take input from one or more elements (streams, windows, or keyed streams), and produce records when a prescribed pattern is found within the input data.

Aggregation [page 193]
An aggregation window collects input records and groups them based on the values in the columns specified with the GROUP BY clause, producing one row of output per group. The output record for each group can contain summary statistics about the records in the group, such as average, sum, count, min, max, etc.

Working with Reference Table Queries [page 196]
Reference table queries enable you to look up information in an SAP HANA database table in response to an incoming event.

Creating and Modifying Compute Queries [page 207]
In the visual editor, produce a simple query that transforms the schema or field values of each incoming record. Each incoming event produces one new output event from the fields defined by the column expressions.

Connecting a Stream to a Derived Window [page 208]
Use a GROUP BY clause or the `nextval()` function to connect a stream to a derived window as part of a complex query.

6.1 Simple Queries

Accomplish most common querying tasks (filter, aggregate, join, compute, union, and pattern) using a set of queries available in the visual editor, or their equivalents in CCL.

The tools for these six queries are available as objects in the palette, in **Streams and Windows**:

Filter	allows you to filter a stream down to only the events of interest, based on a filter expression. Similar to the SQL WHERE clause.
Aggregate	allows you to group events that have common values and compute summary statistics for the group, such as an average. You can also define a window size, based on either time or number of events. Uses the CCL GROUP BY clause, similar to SQL GROUP BY.
Join	allows you to combine records from multiple streams or windows, forming a new record with information from each source. Comparable to a join in SQL, where you specify two or more sources in the FROM clause.
Compute	allows you to create a new event with a different schema, and compute the value to be contained in each column (field) of the new event. Comparable to a projection in SQL, where you use a SELECT statement to specify the column expressions, and FROM to specify a single source.
Union	allows you to combine multiple streams or windows that all share a common schema into a single stream or window. Similar to the SQL UNION operator.

Pattern allows you to watch for patterns of events within a single stream or window or across multiple streams and windows. When smart data streaming detects an event pattern in a running project, it produces an output event. This uses the CCL MATCHING clause.

The following table summarizes the CCL equivalents for simple queries:

Simple Query	CCL
Filter	WHERE clause
Aggregate	GROUP BY clause
Join	FROM clause, WHERE clause, ON clause
Compute	Simple SELECT statement, with column expressions
Union	UNION clause
Pattern	MATCHING clause

Simple Queries from CCL Statements

If you create queries in CCL and want them to appear as simple query shapes in the visual editor, insert a comment immediately preceding the CREATE STREAM or CREATE WINDOW statement, in the following form, where <QUERY-TYPE> is the shape name in the visual editor.

```
/**@SIMPLEQUERY=<QUERY-TYPE>*/
```

For example, this comment causes a CREATE WINDOW statement to map to an aggregate shape in the visual editor: /**@SIMPLEQUERY=AGGREGATE*/ .

Without this comment immediately preceding the CREATE WINDOW statement, the visual editor shows the generic derived window shape.

Note

You cannot modify CCL code in the CCL editor and in the visual editor concurrently. If the visual editor is open, then the CCL editor becomes read-only.

CCL Statements from Simple Queries

When you create a simply query from the palette, the CCL element it creates is based on these rules:

- If the input for the filter, compute, or union object is:
  - A stream, the object creates a stream.
  - A window or Flex stream, the object creates a window.
- If a join object takes input only from streams, then the join object creates a stream. If the input is from one or more windows or Flex streams, then the join object creates a window. In a stream-window join, the join object creates a stream.

- All aggregate objects create a window.
- All pattern objects create a stream.

Related Information

Creating and Modifying Join Queries [page 186]
Creating and Modifying Aggregate Queries [page 195]

6.2 Filtering

Use the WHERE clause in your CCL query to filter data to be processed by the derived elements (streams, windows, or keyed streams).

The WHERE clause restricts the data captured by the SELECT clause, reducing the number of results generated. Only data matching the value specified in the WHERE clause is sent to your derived elements.

The output of your derived element consists of a subset of records from the input. Each input record is evaluated against the filter expression. If a filter expression evaluates to false (0), the record does not become part of the derived element.

When using the WHERE clause with a keyed stream, filter on columns that have values that do not change between an insert event and subsequent update and delete events. If the columns change, related events can be lost; for example, downstream elements might receive update or delete events without the insert that provided the data being updated or deleted, or fail to receive a delete for a previous insert.

This example creates a new window, `IBMTrades`, where its rows are any of the result rows from `Trades` that have the symbol "`IBM`":

```
CREATE WINDOW IBMTrades
    PRIMARY KEY DEDUCED
    AS SELECT * FROM Trades WHERE Symbol = 'IBM';
```

In this section:

Creating and Modifying Filter Queries [page 180]
Use the visual editor to produce a simple query that only passes on events with specific characteristics. Filter uses a CCL WHERE clause.

6.2.1 Creating and Modifying Filter Queries

Use the visual editor to produce a simple query that only passes on events with specific characteristics. Filter uses a CCL WHERE clause.

Procedure

1. In the visual editor palette of the SAP HANA Streaming Development perspective, in **Streams and Windows**, click **Filter** ( ).
2. Select a location in the diagram and click to add the shape.
3. Attach the filter object to any stream, window, or Flex operator. Filter objects can have only one input.
4. Edit the value of the filter expression by selecting the value and changing it as necessary. The default value is 1.
   Any expression that evaluates to '1' is true, and passes all records through. A value of zero is false.
5. (Optional) Use the toggle option to designate the filter object as LOCAL or OUTPUT. By default, filters are OUTPUT.

6.3 Splitting Up Incoming Data

Use the SPLITTER construct to separate incoming data according to filtering rules and write it out to different target streams.

When you want to separate incoming data into several subsets and process those subsets differently, use the CREATE SPLITTER construct. This construct operates like the ANSI case statement: it reads the incoming data, applies the specified filtering conditions, and writes out each subset of the data to one or more target streams.

The target streams are implicitly defined by the compiler. The schema for the target streams are derived based on the `column_list` specification. All the targets are defined as either local or output, depending on the visibility clause defined for the splitter. The default is local.

> **Note**
>
> When the splitter has an output visibility, output adapters can be directly attached to the splitter targets, even though those targets are implicitly defined.

The first condition that evaluates to true (non-zero value) causes the record as projected in the column_list to be inserted into the corresponding target streams. Subsequent conditions are neither considered nor evaluated.

When the source of a splitter is a window:

- Copy the primary keys as-is. The other columns can be changed.

- If you use the output in a join, specify a retention policy.
- The output is a stateless object and does not receive updates, only base data.
- Projections that contain a non-deterministic expression may produce unpredictable results.

Example

This example creates a schema named `TradeSchema` and applies it to the input window Trades. `IBM_MSFT_Splitter` evaluates and routes data to one of three output windows. Event records with the symbol IBM or MSFT are sent to the `IBM_MSFT_Tradeswin` window. Event records where the product of `trw.Price * trw.Volume` is greater than 25,000 are sent to the `Large_TradesWin` window. All event records that do not meet the conditions placed on the two previous output windows are sent to the `Other_Trades` window.

```
CREATE SCHEMA TradeSchema (
Id long,
Symbol STRING,
Price MONEY(4),
Volume INTEGER,
TradeTime SECONDDATE
) ;
CREATE INPUT WINDOW Trades
SCHEMA TradeSchema
PRIMARY KEY (Id) ;
CREATE SPLITTER IBM_MSFT_Splitter
AS
WHEN trw.Symbol IN ('IBM', 'MSFT') THEN IBM_MSFT_Trades
WHEN trw.Price * trw.Volume > 25000 THEN Large_Trades
ELSE Other_Trades
SELECT trw. * FROM Trades trw ;
CREATE OUTPUT WINDOW IBM_MSFT_TradesWin
PRIMARY KEY DEDUCED
AS SELECT * FROM IBM_MSFT_Trades ;
CREATE OUTPUT WINDOW Large_TradesWin
PRIMARY KEY DEDUCED
AS SELECT * FROM Large_Trades ;
CREATE OUTPUT WINDOW Other_TradesWin
PRIMARY KEY DEDUCED
AS SELECT * FROM Other_Trades ;
```

6.4 Unions

Use a UNION operator in your CCL query to combine the results of two or more queries into a single result.

If the union is on a window or keyed stream, duplicate rows are eliminated from the result set due to the primary key. If the union is on a stream, duplicates flow through.

The input for a union operator comes from one or more streams or windows. Its output is a set of records representing the union of the inputs.

Example

This example shows a simple union between two windows, `InStocks` and `InOptions`:

```
CREATE INPUT WINDOW InStocks
```

```
     SCHEMA StocksSchema
     Primary Key (Ts);
CREATE INPUT WINDOW InOptions
     SCHEMA OptionsSchema
     Primary Key (Ts);
CREATE OUTPUT WINDOW Union1
     SCHEMA OptionsSchema
     PRIMARY KEY DEDUCED
     AS SELECT s.Ts as Ts, s.Symbol as StockSymbol,
             Null as OptionSymbol, s.Price as Price, s.Volume as Volume
     FROM InStocks s
UNION
     SELECT s.Ts as Ts, s.StockSymbol as StockSymbol,
             s.OptionSymbol as OptionSymbol,  s.Price as Price,
             s.Volume as Volume
     FROM InOptions s;
```

In this section:

6.4.1 Creating and Modifying Union Queries

Use the visual editor to produce a union query, which combines two or more input streams or windows into a single output. All inputs must have matching schema.

Procedure

1. In the visual editor palette of the SAP HANA Streaming Development perspective, in **Streams and Windows**, select **Union** ( ).
2. Select a location in the diagram and click to add the shape.
3. Attach the union object to two or more inputs, which can be streams, windows, or Flex operators.

> **Note**
>
> To add additional inputs to the union object, you can use the **Connector** tool in the Palette or the **Union** icon ( ) in the shape toolbar.

4. (Optional) Use the toggle option to designate the union object as LOCAL or OUTPUT. By default, unions are OUTPUT.

6.4.2 Example: Merging Data from Streams or Windows

Use the UNION clause to merge data from two streams or windows and produce a derived element (stream, window, or keyed stream).

Procedure

1. Create a new window:

   ```
   CREATE WINDOW <name>
   ```

   You can also create a new stream or keyed stream.
2. Specify the primary key:

   ```
   PRIMARY KEY (…)
   ```

3. Specify the first derived element in the union:

   ```
   SELECT * FROM StreamWindow1
   ```

4. Add the UNION clause:

   ```
   UNION
   ```

5. Specify the second derived element in the union:

   ```
   SELECT * FROM StreamWindow2
   ```

6.5 Joins

Use joins to combine multiple datasources into a single query.

Streams, windows, and keyed streams can participate in a join. A join can contain any number of windows, but only one stream. Self joins are also supported. For example, you can include the same window more than once in a join, provided each instance has its own alias.

In a stream-window join, the target can be a stream or a window with aggregation. Using a window as a target requires an aggregation, because the stream-window join does not have keys and a window requires a key. The GROUP BY column in aggregation automatically forms the key for the target window.

> **Note**
>
> Unnamed windows are implicitly created when using a join with a window that produces a stream. The unnamed window is created to ensure that a join does not see records that have not yet arrived at the join. This can happen because the source to the join and the join itself are running in separate threads.

Joins are performed in pairs, but you can combine multiple joins to produce a complex multitable join. Depending on the complexity and nature of the join, the compiler may create intermediate joins. The comma join syntax supports only inner joins, and the WHERE clause in this syntax is optional. If you omit the WHERE clause, the join creates a many-many relationship between the streams in the FROM clause.

Joins in ANSI syntax can add the DYNAMIC modifier to a window or stream to indicate that its data changes frequently. A secondary index is created on windows joining with an incomplete primary key of a DYNAMIC window or stream. This improves performance but uses additional memory proportional to the total data length of key columns in the index. By default, windows and streams are STATIC and no secondary indexes are created.

The following table describes the join types supported by smart data streaming.

Join Type	Syntax	Description
Inner Join	`INNER JOIN`	One record from each side of the join is required for the join to produce a record.
Left Outer Join	`LEFT JOIN`	A record from the left side (outer side) of the join is produced regardless of whether a record exists on the right side (inner side). When a record on the right side does not exist, any column from the inner side has a NULL value.
Right Outer Join	`RIGHT JOIN`	Reverse of left outer join, where the right side is the outer side and the left side is the inner side of the join.
Full Outer Join	`FULL JOIN`	A record is produced whether there is a match on the right side or the left side of the join.

The following table describes the join cardinalities supported by smart data streaming.

Type	Description
One-One	Keys of one side of the join are completely mapped to the keys of the other side of the join. One incoming row produces only one row as output.
One-Many	One record from the one side joins with multiple records on the many side. The one side of the join is the side where all the primary keys are mapped to the other side of the join. Whenever a record comes on the one-side of the join, it produces many rows as the output.
Many-Many	The keys of both side of the join are not completely mapped to the keys of the other side of the join. A row arriving on either side of the join has the potential to produce multiple rows as output.

Note

When a join produces multiple rows, the rows are grouped into a single transaction. If the transaction fails, all of the rows are discarded.

Example

This example joins two windows (`InStocks` and `InOptions`) using the FROM clause with ANSI syntax. The result is an output window:

```
CREATE INPUT Window InStocks SCHEMA StocksSchema Primary Key (Ts) ;
CREATE INPUT Window InOptions SCHEMA OptionsSchema Primary Key (Ts) KEEP ALL ;
CREATE Output Window OutStockOption  SCHEMA OutSchema
    Primary Key (Ts)
    KEEP ALL
AS
    SELECT InStocks.Ts Ts,
        InStocks.Symbol Symbol,
        InStocks.Price StockPrice,
        InStocks.Volume StockVolume,
        InOptions.StockSymbol StockSymbol,
```

```
        InOptions.OptionSymbol OptionSymbol,
        InOptions.Price OptionPrice,
        InOptions.Volume OptionVolume
    FROM InStocks JOIN InOptions
      ON
        InStocks.Symbol = InOptions.StockSymbol and
                    InStocks.Ts = InOptions.Ts ;
```

Keyed Stream Joins

When a keyed stream is the target of an inner join, ensure that the columns on which the join is performed do not change across an insert event and related update and delete events that follow. If the columns change, related events can be lost; the keyed stream may send update or delete events without the insert that provided the data being updated or deleted, or fail to send a delete for a previous insert.

Keyed streams are stateless except when performing a join with a window. In this type of join, the keyed stream uses memory to store a reference to the records in the window.

Restrictions on joins:

- A keyed stream can only be an outer member of an outer join (inner joins are supported).
- A keyed stream cannot participate in a full join.
- When you join a keyed stream with a window, only events that arrive in the keyed stream trigger a join; changes to the window do not.
- A keyed stream cannot be the target of a join when all inputs to the join are windows (named or unnamed).

Nested Joins

Implementing a nested join requires several important functions. Nested join syntax is supported in CCL, but you cannot create or edit a nested join in the visual editor. When a nested join is defined in the CCL file, and you switch to the visual editor, you see an empty join compartment.

In this section:

6.5.1 Creating and Modifying Join Queries

Using the visual editor, produce a simple query that combines fields from multiple input events into a single output event.

Procedure

1. In the visual editor palette, in **Streams and Windows**, select **Join** ().
2. Select a location in the diagram and click to add the shape.
3. Connect the join object to the streams or windows that provide the inputs to the join.

   Connect join objects to two or more streams, windows, or Flex operators. Join objects can take input from two or more objects, but can produce only one output.

   > Note
   >
   > Streams, windows and keyed streams can participate in a join. However, only one stream can participate in a join. For details of supported joins, see Joins [page 183].

   > Tip
   >
   > To add multiple connections, **Shift+click** while holding the **Connector** tool. To return to normal selection, press **Esc** or click the **Select** tool in the palette to release it.

4. Use **Copy Columns from Input** to select input fields to include in the output of this query.
5. Add column expressions, as necessary.
6. Edit a column expression by selecting one of the tabs in the Properties view and editing the corresponding fields, or by:
   - Double-clicking to open the inline editor.
   - Selecting an expression and pressing **Ctrl+F2** to edit it using the pop-up expression editor.

   > Tip
   >
   > When entering column names and their datatypes, use **Tab** to easily move between cells in the table.

7. Click **Add Join Condition** to specify the columns to use to match incoming events across the different sources.

   Complete the **Edit Join Expression** dialog to define the join type, data sources for the ON clause, and any other join constraints.

   > Note
   >
   > If you create a join using comma-separated syntax in the CCL editor, and then add an ON clause using the **Edit Join Expression** dialog in the visual editor, the WHERE clause initially created in the comma-separated syntax will not be removed. This does not affect the result, however it will negatively affect performance.

If you do not see the columns you want in the **Edit Join Expression** dialog, ensure you have connected the join object to the correct input sources.

8. To join a column to itself, click **Add Input Alias** in the shape toolbar.
   A column alias is required to provide a unique name for each join condition.
9. (Optional) Use the toggle option to designate the join object as LOCAL or OUTPUT. By default, joins are OUTPUT.
10. (Optional) Select **Set Keep Policy** and choose an option.
    To edit the keep policy, right-click the input window or stream in the **Inputs** menu. Select **Set Keep Policy** to add a keep policy, and **Delete Keep Policy** to remove it.

Related Information

Specifying a Retention Policy [page 87]
Simple Queries [page 177]

6.5.2 Key Field Rules

Key field rules ensure that rows are not rejected due to duplicate inserts or the key fields being NULL.

- The key fields of the target are always derived completely from the keys of the 'many' side of the join. In a many-many relationship, the keys are derived from the keys of both sides of the join.
- In a one-one relationship, the keys are derived completely from either side of the relationship.
- In an outer join, the key fields are derived from the outer side of the join. An error is generated if the outer side of the join is not the many-side of a relationship.
- In a full-outer join, the number of key columns and the type of key columns need to be identical in all sources and targets. Also, the key columns require a `firstnonnull` expression that includes the corresponding key columns in the sources.

When the result of a join is a window, specific rules determine the columns that form the primary key of the target window. In a multitable join, the same rules apply because conceptually each join is produced in pairs, and the result of a join is then joined with another stream or window, and so on.

The following table summarizes the key field rules for joins:

Join Type	One-One	One-Many	Many-One	Many-Many
INNER	Include keys from at least one side in the projection list (or a combination of them if keys are composite).	Include keys from the right side in the projection list.	Include keys from the left side in the projection list.	Include keys from both sides in the projection list.
LEFT	Include keys from the left side alone.	Not allowed.	Include keys from the left side in the projection list.	Not allowed.

Join Type	One-One	One-Many	Many-One	Many-Many
RIGHT	Include keys from the right side alone.	Include keys from the right side in the projection list.	Not allowed.	Not allowed.
OUTER	Form keys using `firstnonnull()` on each pair of keys from both sides.	Not allowed.	Not allowed.	Not allowed.

6.5.3 Join Examples: ANSI Syntax

Examples of different join types using the ANSI syntax.

Refer to these inputs for the examples below:

```
CREATE INPUT STREAM S1 SCHEMA (Val1S1 integer, Val2S1 integer, Val3S1 string);
CREATE INPUT WINDOW W1 SCHEMA (Key1W1 integer, Key2W1 string, Val1W1 integer,
Val2W1 string) PRIMARY KEY (Key1W1, Key2W1);
CREATE INPUT WINDOW W2 SCHEMA (Key1W2 integer,  Key2W2 string, Val1W2 integer,
Val2W2 string) PRIMARY KEY (Key1W2, Key2W2);
CREATE INPUT WINDOW W3 SCHEMA (Key1W3 integer,  Val1W3 integer, Val2W3 string)
PRIMARY KEY (Key1W3);
```

Simple Inner Join: One-One

Here, keys can be derived from either W1 or W2:

```
CREATE OUTPUT WINDOW OW1
PRIMARY KEY  (Key1W2, Key2W2)
AS SELECT W1.*, W2.*
FROM W1 INNER JOIN W2 ON W1.Key1W1 = W2.Key1W2 AND W1.Key2W1 = W2.Key2W2;
```

Simple Left Join: One-One

The keys are derived from the outer side of the left join. It is incorrect to derive the keys from the inner side because the values could be null:

```
CREATE OUTPUT WINDOW OW2
PRIMARY KEY (Key1W1, Key2W1)
AS SELECT W1.*, W2.*
FROM W1 LEFT JOIN W2 ON W1.Key1W1 = W2.Key1W2 AND W1.Key2W1 = W2.Key2W2;
```

Simple Full Outer Join: One-One

The key columns all have a required `firstnonnull` expression in it:

```
CREATE OUTPUT WINDOW OW3
PRIMARY KEY (Key1, Key2)
AS SELECT firstnonnull(W1.Key1W1, W2.Key1W2) Key1, firstnonnull(W1.Key2W1,
W2.Key2W2) Key2, W1.*, W2.*
FROM W1 FULL JOIN W2 ON W1.Key1W1 = W2.Key1W2 AND W1.Key2W1 = W2.Key2W2;
```

Simple Left Join: Many-One

All the keys of W2 are mapped and only one key of W1 is mapped in this join. The many-side is W1 and the one-side is W2. The keys are derived from the many-side:

```
CREATE OUTPUT WINDOW OW4
PRIMARY KEY (Key1W1, Key2W1)
AS SELECT W1.*, W2.*
FROM W1 LEFT JOIN W2 ON W1.Key1W1 = W2.Key1W2 AND W1.Val2W1 = W2.Key2W2;
```

Simple Left Join: Many-One (DYNAMIC Modifier)

W3 is `DYNAMIC` and only one key of W1 is mapped in this join, so a secondary index is created on W1. W1 is also `DYNAMIC`, but all keys of W3 are mapped, so no secondary index is created on W3:

```
CREATE OUTPUT WINDOW OW5
PRIMARY KEY DEDUCED
AS SELECT W1.*, W3.*
FROM W1 (DYNAMIC) LEFT JOIN W3 (DYNAMIC) ON W1.Key1W1 = W3.Key1W3;
```

Simple Inner Join: Many-Many

This is a many-many join because neither of the keys is fully mapped. The keys of the target are the keys of all the windows participating in the join:

```
CREATE OUTPUT WINDOW OW6
PRIMARY KEY (Key1W1, Key2W1, Key1W2, Key2W2)
AS SELECT W1.*, W2.*
FROM W1 JOIN W2 ON W1.Val1W1 = W2.Val1W2 AND W1.Val2W1 = W2.Val2W2;
```

Simple Stream-Window Left Join

When a left join involves a stream, the stream is on the outer side. The target cannot be a window unless it is also performing aggregation:

```
CREATE OUTPUT STREAM OSW1
AS SELECT S1.*, W2.*
FROM S1 LEFT JOIN W2 ON S1.Val1S1 = W2.Key1W2 AND S1.Val3S1 = W2.Key2W2;
```

Complex Window-Window Join

The keys for OW4 can be derived either from W1 or W2 because of the inner join between the two tables:

```
CREATE OUTPUT WINDOW OW7
PRIMARY KEY DEDUCED
AS SELECT S1.*, W1.*, W2.*, W3.*   //Some column expression.
FROM S1 LEFT JOIN (W1 INNER JOIN (W2 LEFT JOIN W3 ON W2.Key1W2 = W3.Key1W3) ON
W1.Key1W1 = W2.Key1W2 AND W1.Key2W1 = W2.Key2W2) ON S1.Val1S1 = W1.Key1W1
WHERE W2.Key2W2 = 'abcd'
GROUP BY W1.Key1W1, W2.Key2W2
HAVING SUM(W3.Val1W3) > 10;
```

Complex Stream-Window Join

Here, the join is triggered only when a record arrives on S1. Also, because there is aggregation, the target is a window instead of being restricted to a stream:

```
CREATE OUTPUT WINDOW OW8
PRIMARY KEY DEDUCED
AS SELECT S1.*, W1.*, W2.*, W3.*   //Some column expression.
FROM S1 LEFT JOIN (W1 INNER JOIN (W2 LEFT JOIN W3 ON W2.Key1W2 = W3.Key1W3) ON
W1.Key1W1 = W2.Key1W2 AND W1.Key2W1 = W2.Key2W2) ON S1.Val1S1 = W1.Key1W1
WHERE W2.Key2W2 = 'abcd'
GROUP BY W1.Key1W1, W2.Key2W2
HAVING SUM(W3.Val1W3) > 10;
```

6.5.4 Join Example: Comma-Separated Syntax

An example of a complex join using the comma-separated syntax.

This join is a complex join of three windows using the comma-separated join syntax. The WHERE clause specifies the conditions on which records are joined.

```
CREATE OUTPUT WINDOW OW4
PRIMARY KEY DEDUCED
AS SELECT W1.*, W2.*, W3.*
FROM W1, W2, W3
WHERE W1.Key1W1 = W2.Key1W2 AND W1.Key2W1 = W2.Key2W2  AND W1.Key1W1 = W3.Key1W3;
```

6.6 Pattern Matching

Use the MATCHING clause in your CCL query to take input from one or more elements (streams, windows, or keyed streams), and produce records when a prescribed pattern is found within the input data.

Pattern streams can check whether events (rows from the input sources matching certain criteria) occur during a specific time interval, then send records to downstream streams if a match has occurred.

Pattern matching can be used to distill complex relationships between data into compact and easily-maintainable expressions.

> **Note**
>
> The pattern rule engine uses any incoming event in order to match the defined pattern, regardless of the opcode of the incoming event. Include the opcode in each event's definition to filter out unwanted rows.

This example creates an output stream, `ThreeTrades`, which monitors the `QTrades` streams and sends a new event when it detects three trades on the same symbol within five seconds. The output of this stream is the symbol of the traded stock, and its latest three prices. The trades do not have to occur consecutively, but must occur within five seconds of each other. Multiple patterns may be in the process of being matched at the same time.

```
CREATE OUTPUT STREAM ThreeTrades
AS
SELECT
    T1.Symbol,
    T1.Price Price1,
    T2.Price Price2,
    T3.Price Price3
FROM
    QTrades T1,
    QTrades T2,
    QTrades T3
MATCHING[5 SECONDS: T1, T2, T3]
ON T1.Symbol = T2.Symbol = T3.Symbol;
```

For details on the MATCHING clause, see the *SAP HANA Smart Data Streaming: CCL Reference Guide.*

In this section:

Related Information

SAP HANA Smart Data Streaming: CCL Reference

6.6.1 Creating and Modifying Pattern Queries

In the visual editor, run a pattern-matching query that watches for a specific pattern of incoming events on one or more inputs and produces an output event when the pattern is detected. Pattern uses the CCL MATCHING clause.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor palette, in **Streams and Windows**, click **Pattern** ().
3. Select a location in the diagram and click to add the shape.
4. Connect the Pattern shape to one or more streams or windows that are the inputs to query.
5. Add columns:
    a. Click **Copy Columns from Input** () in the shape toolbar to select the columns to copy into the schema for the Pattern query.

       This is the schema of the new event that is produced when the pattern is detected.
    b. Add additional columns by clicking **Add Column Expression** in the shape toolbar.
6. Edit a column expression by selecting one of the tabs in the Properties view and editing the corresponding fields, or by:
    - Double-clicking to open the inline editor.
    - Selecting an expression and pressing **Ctrl+F2** to edit it using the pop-up expression editor.

    > Tip
    >
    > When entering column names and their datatypes, use **Tab** to easily move between cells in the table.
7. Create and edit a pattern expression:
    a. Click **Add Pattern** ().
    b. Enter an alias for the event.
    c. Enter either a time interval or parameters.
    d. To define the expression, right-click **Pattern** to add an event. Continue right-clicking elements of the expression to add operators and refine the event expression. Then click **Next.**
    e. Click **Add** to add a join condition.
8. (Optional) Use the toggle option to designate the pattern object as LOCAL or OUTPUT. By default, patterns are OUTPUT.

6.7 Aggregation

An aggregation window collects input records and groups them based on the values in the columns specified with the GROUP BY clause, producing one row of output per group. The output record for each group can contain summary statistics about the records in the group, such as average, sum, count, min, max, etc.

All records in each group have the same values for the columns specified in the GROUP BY clause. All columns specified in the GROUP BY clause must also be included in the SELECT clause, because these columns form the primary key for the aggregation window. This is why the primary key for the aggregate window uses the PRIMARY KEY DEDUCED clause instead of explicitly specifying a primary key.

When using aggregation, consider the memory usage implications. All of the input records for which an aggregate function is to be calculated are stored in memory. The data structure that holds all the records in memory is called the aggregation index.

Using the KEEP Clause for Aggregation Windows

When creating a window in CCL, the KEEP policy can be specified as part of the FROM clause to define an unnamed window, or it can be defined for the window itself. In the case of an aggregation window, the effects of the two are very different and are often misunderstood.

The first (and most common) method is to apply a KEEP clause to the input, creating an unnamed window that will be used for the input to the aggregation. In the following example, the aggregation is applied to an unnamed window that contains all events received from stream `Stream123` in the last 5 minutes:

```
CREATE INPUT STREAM Stream123
SCHEMA (SourceID string ,
        Value Integer ,
        Tstamp msdate );
/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW MovingAverage
PRIMARY KEY DEDUCED
AS SELECT
        Stream1.SourceID SourceID ,
        avg ( Stream123.Value ) Value ,
        max ( Stream123.Tstamp ) Tstamp
FROM Stream123 KEEP 5 MINUTES
GROUP BY Stream123.SourceID ;
```

The second method is to apply a KEEP clause to the window being created. Here, the KEEP policy will be applied to the output; it will be applied after aggregation and will be applied to the group records. Thus the KEEP policy will delete the summary record of any group that hasn't been added or updated recently, according to the KEEP policy. The following example uses this method:

```
CREATE INPUT WINDOW Window1
SCHEMA (SourceID string ,
        Value Integer ,
        Tstamp msdate )
PRIMARY KEY (SourceID)
KEEP 1 HOUR;
/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW MovingAverage
PRIMARY KEY DEDUCED
KEEP 5 MINUTES
```

```
AS      SELECT
         Window1.SourceID SourceID ,
         avg ( Window1.Value ) Value ,
         max ( Window1.Tstamp ) Tstamp
FROM Window1
GROUP BY Window1.SourceID ;
```

Note that this example does not create a 5 minute moving average. The KEEP 5 MINUTES policy is applied after aggregation and is applied to the rows in the window `MovingAverage`. So the average and max values will be computed across all records held in `Window1` (which has all record received in the last 1 hour), but any group that has not been added or updated in the last 5 minutes will be deleted from the `MovingAverage` window. Thus the window `MovingAverage` has a one hour moving average for all groups added or updated in the last 5 minutes.

Aggregating Over Streams: Avoiding Unbounded Growth

You often need to create an aggregation window that is fed by a stream, allowing you to compute aggregate values over events received from the stream. To compute aggregation values, an aggregation index is created on the input to an aggregation window. When a stream serves as the input to an aggregation window, it's important to specify a KEEP policy as part of the FROM clause, to limit the size of the aggregation index. Failure to specify a KEEP policy in the FROM clause will lead to unbounded growth of the aggregation index, eventually consuming all system resources and causing the project to shut down.

To prevent this, construct a SELECT clause that only uses additive aggregation functions. These are special aggregate functions that do not require an aggregation index and are computed incrementally.

When the projection contains only additive aggregation functions, the server performs additive optimization; therefore, there is no need to maintain an aggregation index, and memory does not grow. Supported additive aggregation functions are `sum`, `count`, `avg`, and `valueInserted`. For more information, see Improving Aggregation Performance [page 343]. Note that `last()` is not an additive function; you must use `valueinserted()` rather than `last()`. Here is an example:

```
CREATE INPUT STREAM Stream2
SCHEMA (SourceID string ,
         Value Integer ,
         Tstamp msdate );
/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW MovingAverage
PRIMARY KEY DEDUCED
AS      SELECT
         Stream2.SourceID SourceID ,
         avg ( Stream2.Value ) Value ,
         valueInserted ( Stream2.Tstamp ) Tstamp
FROM Stream2
GROUP BY Stream2.SourceID ;
```

> **i Note**
>
> If even one of the non-GROUP BY column expressions does not have an additive aggregation function, the server needs to maintain an aggregation index resulting in unbounded memory growth. If you select a column without specifying an additive aggregation function, it is considered as a non-additive function forcing the server to maintain an aggregation index.

Advanced Aggregation Techniques

In addition to the GROUP BY clause, you can optionally specify a GROUP FILTER and GROUP ORDER BY clause. The GROUP ORDER BY clause orders the records in a group by the specified columns before applying the GROUP FILTER clause and the aggregation functions. With the records ordered, aggregation functions sensitive to the order of the records such as `first`, `last`, and `nth` can be used meaningfully.

The GROUP FILTER clause is executed after the GROUP ORDER BY clause and eliminates any rows in the group that do not meet the filter condition. The specified filter condition can be used independently of the GROUP ORDER BY clause, but by ordering the group before filtering, the rank function can be used to filter the group. Here is an example:

After you execute the GROUP ORDER BY clause, every row in the group is ranked from 1 to N. This allows you to specify `rank () < 11` in the GROUP FILTER clause, which means that the aggregation function is only applied to the first 10 rows in the group after it has been ordered by the columns specified in the GROUP ORDER BY clause.

You can also specify an optional HAVING clause, which applies a filter after applying the aggregation functions to the records in a group. In other words, a filter specified in a HAVING clause will filter out the aggregate record for any group that does not meet the filter criteria.

> **Note**
>
> Note: You can only specify GROUP ORDER BY, GROUP FILTER, and HAVING clauses in conjunction with a GROUP BY clause.

In this section:

6.7.1 Creating and Modifying Aggregate Queries

Using the visual editor, produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor palette, in **Streams and Windows**, select **Aggregate** ().
3. Select a location in the diagram and click to add the shape.
4. Connect the Aggregate shape to an input.
   The aggregate border changes from red to black, indicating that it is valid, now that it has input.

5. Add columns:
   a. Click **Copy Columns from Input** ( ) in the shape toolbar to select the columns to copy into the schema for the Aggregate window.
   b. Add additional columns by clicking **Add Column Expression** in the shape toolbar.
6. Edit a column expression by selecting one of the tabs in the Properties view and editing the corresponding fields, or by:
   - Double-clicking to open the inline editor.
   - Selecting an expression and pressing **Ctrl+F2** to edit it using the pop-up expression editor.

   > Tip
   >
   > When entering column names and their datatypes, use **Tab** to easily move between cells in the table.
7. Click **Add GroupBy Clause** ({ }) in the shape toolbar to edit the grouping of columns in the aggregate object.

   > Note
   >
   > The Aggregate shape must have exactly one GROUP BY expression.
8. (Optional) Click **Set Keep Policy** ( ) to create a retention window.

   The default policy is to keep all rows of incoming data. You can also choose to keep only the last row, a specific number of rows, or keep the rows for a specific time. This defines the KEEP clause. You can also go further, and retain the rows defined by the KEEP clause to span retention across multiple specified columns. This spanning of retention across columns is done by listing column names in the PER clause.
9. (Optional) Use the Toggle option to designate the aggregate object as LOCAL or OUTPUT. By default, aggregates are OUTPUT.

Related Information

Specifying a Retention Policy [page 87]
Simple Queries [page 177]

6.8 Working with Reference Table Queries

Reference table queries enable you to look up information in an SAP HANA database table in response to an incoming event.

Incoming events can arrive with only a portion of the information necessary to complete the processing you wish to specify in your project. When that additional information is present in existing tables in an SAP HANA database, you can use reference table queries to look it up. There are two parts to this: creating the table reference query, and using it to execute an ad hoc query in a join or Flex operator.

When creating the reference, determine what data you want to use. Then identify the table containing the data by name, obtain the schema of the table, and find out what service to use to contact the database. Decide whether you want to attempt to reconnect if the connection is dropped, and if so, how many times, and how long to wait between attempts.

When joining a stream or window to a reference, determine what you want as the output of the join. There are numerous restrictions on how to use references in joins, and what types of output you can obtain. For example, if you want the output of the join to be a window, you have to specify the primary key of the reference and use the complete primary key in the ON or WHERE clause of the join.

There are several different ways to use references within CCLScript programs. You can iterate over the rows in the table or grab specific rows. Basically, you can use a reference in the same ways you use a window. It is simply another source of data for processing in your CCLScript routine.

You can use references—in joins and in CCLScript programs—inside a module, as well as within the main body of your project. Like stores, references used in a module must be bound to a reference defined in the main body of your project.

Prerequisites

Install the SAP HANA ODBC client on the system where you want to run projects that include reference table queries. The minimum required version of the SAP HANA ODBC client is 1.0.73, but you should install the most recent version.

Database Service Definition

All connections to external databases, including reference table queries, are made using data services defined in the cluster. You can define or modify a data service definition using the Data Services view in the SAP HANA Streaming Development perspective. You can also use studio to define the default SAP HANA service entry.

> Note
>
> If the SAP HANA database on which you are making these queries is on an SAP HANA cluster, see *Configuring to Support SAP HANA Failover* in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for details on how to add the other nodes in the SAP HANA cluster to the database service definition.

Caching

When a project joins streaming data to SAP HANA tables, such as customer or instrument information, reference table queries may repeatedly make the same requests. Since turning on caching for a reference table query enables local storage of previous query results, caching can improve the performance of the join or Flex operation using the reference table query. It can also reduce network traffic when the SAP HANA table being queried is on the network.

By default, caching is turned off (you can explicitly turn it off by setting `cachePolicy` to **`NONE`**). Turn it on by setting `cachePolicy` to **`ONACCESS`**.

By default, there are no limits on the size of the cache or the age of the cached query results, so the size of the cache keeps increasing as query results are cached. And, once the query results have been retrieved and cached, they will never be retrieved again.

If you want to be able to retrieve query results, you can specify the maximum physical size of the cache (`maxCacheSizeMB`), or how old a cached query result can be (`maxCacheAge`) before it is no longer returned when the query runs again, or both.

Error Handling

When you start a project that contains a reference table query, the query does a table schema compatibility check. The reference scans the database table and verifies that:

- For each column specified in the reference, there is a column of the same name (case-insensitive) in the table.
- The datatype of the column in the table is compatible with the datatype of the column in the reference.
- If the reference definition specifies a primary key, there is a matching primary key in the database. If the reference definition doesn't specify a primary key, it doesn't matter whether or not the database has a primary key.

To check the type for each mapped column and to be as database-agnostic as possible, the reference attempts to pull a sample row from the database. If it can pull that column into smart data streaming, the check succeeds. Otherwise it fails, except in these cases:

- If the query that the reference uses to do the type-checking is rejected by the database (because it doesn't support SQL 2003 standards), the reference doesn't complete type-checking, but allows the project to start up, providing a warning that it can't guarantee that the type mapping is valid.
- If the table has no data in it, the type-checking stops, and a warning is printed that it can't guarantee that the type mapping is valid.

While a project is running, the error scenarios are mostly connection-based. When a failure is caused by a lost connection, the server attempts to reconnect based on the reconnect parameters specified in the reference's definition.

In this section:

Use a reference table query to obtain data from an SAP HANA database table for processing by a CCLScript routine in a Flex operator.

Using a Reference Table Query in CCLScript [page 204]
: This example uses a reference table query within a CCLScript routine that iterates over the rows in a table to obtain data about a specified baseball player's batting history that is then used to compute statistics.

Using a Reference Table Query in CCLScript to Get a Single Row from a Table [page 206]
: This example uses a reference table query within a CCLScript routine to obtain a single row from a table.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide

6.8.1 Creating a Reference Table Query Using Schema Discovery

Create a reference in a smart data streaming project that queries a table in an SAP HANA database.

Context

You can create reference table queries either in the studio visual editor, or directly in CCL. Create a reference table query in CCL using the CREATE REFERENCE statement.

If you create the reference table query using the visual editor, you can:

- Use the schema discovery wizard to fill out the query properties.
- Manually enter values for each property.
- Use a mix of both discovery and manual input through the properties view.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the palette, expand Streams and Windows and select **Reference** ( ).
3. Select a location in the diagram and click to add the shape.
4. Provide a name for the **Reference** ( ).
5. Click on Schema Discovery ( ).

Studio displays the Schema Discovery dialog box. The **service** field is populated with the name of the default SAP HANA service entry, as defined in studio. If you have not designated a default SAP HANA service entry, you are prompted to do so.

6. In the **service** field:
   - Leave the specified service name, to use it when connecting to the external database.
   - Replace the specified service with another service, as defined in the studio Data Services View.
   - Click on **Discover** and select from the list of services displayed.
7. In the **Source Schema** field:
   - Enter the name of the schema for the external database containing the table the reference will query.
   - Click on **Discover** and select from the list of schemas displayed.
8. In the **Source** field:
   - Enter the name of the table the reference will query.
   - Click on **Discover** and select from the list of tables displayed.
9. Select the **Discover Primary Keys** checkbox to define primary keys in the reference matching those in the external database table. This is optional, but highly recommended to provide maximum flexibility when using the reference.
10. Click **OK**.

    By default, smart data streaming builds the reference schema based on the schema of the external database table. Once the reference schema has been built, you can remove a column by right-clicking anywhere on the column and choosing **delete element**. Edit the column properties within the reference shape by double-clicking on the property names or edit them in the Properties view. The datatypes in the source schema for the reference must be compatible with those in the target SAP HANA database table.

    You can create a reference within a module; a reference created outside of a module cannot be used in a module. See Creating a Module in the Visual Editor [page 148] .

Results

This method omits several additional properties that you can set through the properties view of the SAP HANA Streaming Development perspective. See *CREATE REFERENCE Statement* in the *SAP HANA Smart Data Streaming: CCL Reference* for a list of all the properties and their definitions.

Related Information

SAP HANA Smart Data Streaming: CCL Reference

6.8.2 Using a Reference Table Query in a Join

This example shows the procedure for creating a reference table query, an input stream, and an output stream that implements a join of data from the reference table query and the input stream to add customer data to an incoming order.

Procedure

1. Obtain the necessary information about the external database table containing the information you wish to look up:
   - The name of the table containing the data
   - The schema for that table
   - The service to use to connect to it
2. Create the reference table query in your project:

   ```
   CREATE REFERENCE customerRef
   SCHEMA (customerID integer, fullName string, address string)
   PRIMARY KEY (customerID)
   PROPERTIES
   service='databaseLookup',source='customerAddr',sourceSchema='addressSchema';
   ```

   The schema of the reference does not have to include all of the columns in the source table or list them in the same order. For each column that you do specify in the schema, however, the column name must match the column name in the database, and the datatype must be compatible with the datatype in the database.

   The `customerID` column is the primary key in the source table, and it contains a unique number assigned to each customer. A primary key is not required to produce a stream; only a window. If you declare a primary key for the reference, however, it must match the primary key in the table. So, whenever the table has a primary key, it is a good practice to make it the primary key of the reference table query (giving you the flexibility to produce either a stream or a window).

   When you specify a primary key in a reference table query, also include the `sourceSchema` property, which identifies the schema, in this case `addressSchema`, of the database containing the table.
3. Create the input stream for events to enter the project:

   ```
   CREATE INPUT STREAM orderStream
   SCHEMA (orderID integer, customerID integer, itemID integer);
   ```

4. Create an output stream that implements a join of the incoming event with the data obtained from the reference table query:

   ```
   CREATE OUTPUT STREAM orderWithCustomerInfoStream
   SCHEMA (orderID integer, customerName string, customerAddress string)
   AS SELECT orderStream.orderID, customerRef.fullName, customerRef.address
   FROM orderStream, customerRef
   WHERE orderStream.customerID=customerRef.customerID;
   ```

   While the column names in the output stream's schema declaration, `customerName` and `customerAddress`, do not have to match the column names in the table, `fullName` and `address`, the datatypes in the output stream's schema must match the datatypes in the table.

6.8.3 Using a Reference Table Query in a Module

This example obtains a bibliography for the specified author using a reference table query inside of a module.

Context

In this example, an order for a book produces a list of other books by that author. Because this list may be desired in response to other events, it is produced within a module.

Procedure

1. Obtain the necessary information about the external database table containing the information you wish to look up:
   - The name of the table containing the data
   - The schema for that table
   - The service to use to connect to it
2. Create the reference table query in your project. When you define a reference in a module, it is always mapped to a reference in your project that contains the connection information for the database:

   ```
   CREATE REFERENCE getBiblio
   SCHEMA (authorName string, bookTitle string, publisher string)
   PROPERTIES
   service='databaseLookup',source='bibliographies',sourceSchema='bookSchema';
   ```

   If you think that the reference table query will make the same requests many times, specify a cache. This example shows a cache that can use up to two megabytes of storage and holds a query result for up to two hours:

   ```
   CREATE REFERENCE getBiblio
   SCHEMA (authorName string, bookTitle string, publisher string)
   PROPERTIES
   service='databaseLookup',source='bibliographies',sourceSchema='bookSchema',
   cachePolicy='ONACCESS',maxCacheSizeMB=2,maxCacheAge=2 hours;
   ```

   > Note
   >
   > To decide whether to enable caching, consider how often you expect the same request to be made. To decide whether to limit the size of the cache or the amount of time a query result is held, consider the amount and type of data you expect the query to return.

3. Create an input stream for order events to enter the project:

   ```
   CREATE INPUT STREAM orderStream
   SCHEMA (orderID integer, customerID integer, itemID integer, authorName
   string);
   ```

4. Create a module that uses this table reference query. In the module, you only need to specify the name and schema of the reference table query:

```
CREATE MODULE listOtherBooks
IN orderStream
OUT otherBooks
REFERENCES getBiblio
BEGIN
    CREATE REFERENCE getBiblio
        SCHEMA (authorName string, bookTitle string, publisher string);
    CREATE INPUT STREAM orderStream
        SCHEMA (orderID integer, customerID integer, itemID integer,
authorName string);
    CREATE OUTPUT STREAM otherBooks
        SCHEMA (authorName string, bookTitle string, publisher string)
        AS SELECT orderStream.authorName, getBiblio.bookTitle,
getBiblio.publisher
        FROM orderStream, getBiblio
        WHERE orderStream.authorName=getBiblio.authorName;
END;
```

5. Load the module to generate the list of other works by the author of the book that was ordered. Loading the module creates a mapping between the streams in the main project and in the module. The output from the module goes to the `otherBooks` local stream in the main project:

```
LOAD MODULE listOtherBooks AS listAll
    IN
        orderStream = orderStream
    OUT
        otherBooks = otherBooks
    REFERENCES
        getBiblio = getBiblio;
```

6. Create an output stream to read the local stream `otherBooks` and display the list of other books by the author:

```
CREATE OUTPUT STREAM otherBooksOutput
SCHEMA (authorName string, bookTitle string, publisher string)
AS SELECT * FROM otherBooks;
```

6.8.4 Using a Reference Table Query in a Flex Operator

Use a reference table query to obtain data from an SAP HANA database table for processing by a CCLScript routine in a Flex operator.

Procedure

1. Obtain the necessary information about the external database table containing the information you wish to look up:
   - The name of the table containing the data
   - The schema for that table
   - The service to use to connect to it

2. Create the reference table query in your project.
3. Create the input stream or window.
4. Create the Flex operator:
   a. From the palette, under **Streams and Windows**, select **Flex** ( ), then click an empty area in the diagram.
   b. From the palette, select **Connector** and connect the input stream or window to the Flex operator.
   c. From the palette, select **Connector** and connect the reference table query to the Flex operator.
   d. Define the schema for the Flex operator.
   e. (Optional) Click on **Aging Policy** ( ).
   f. (Optional) Click **Set Output Keep Policy** ( ) to set keep policy options.
5. Implement a CCLScript routine:
   a. In the Flex operator, under Methods, click on the ( ) of the expression you wish to edit.
   b. In the **Edit Expression Value** dialog box, write the CCLScript script you wish to implement.

      See Using a Reference Table Query in CCLScript [page 204] and Using a Reference Table Query in CCLScript to Get a Single Row from a Table [page 206] for examples of CCLScript routines in a Flex operator.

6.8.5 Using a Reference Table Query in CCLScript

This example uses a reference table query within a CCLScript routine that iterates over the rows in a table to obtain data about a specified baseball player's batting history that is then used to compute statistics.

Procedure

1. Obtain the necessary information about the external database table containing the information you wish to look up:
   - The name of the table containing the data.
   - The schema for that table.
   - The service to use to connect to it.
2. Create the reference table query in your project.

   For example, create a reference table query named `AtBats`:

```
CREATE REFERENCE AtBats
SCHEMA (teamAtBat INTEGER, player STRING, bases INTEGER)
PRIMARY KEY (teamAtBat)
PROPERTIES service='HANA',source='Batting',sourceSchema='BattingSchema';
```

   The `teamAtBat` is incremented each time a player bats a ball, providing a unique number for each at-bat by a member of the team, so that column is the primary key in the source table. It must also be the primary key of the reference table query. Because we have specified a primary key in this reference, we have to include the `sourceSchema` property, which identifies the schema, `BattingSchema`, of the database containing the table.

3. Create the input stream for events to enter the project.

   For any event that is the appearance, or mention, of a player's name, create an input stream named StatRequest, consisting of a single string, player:

```
CREATE INPUT STREAM StatRequest SCHEMA (player STRING);
```

4. Initialize the Flex operator, named Percentage, to read in the reference query AtBats and the input stream StatRequest, specify the output stream PercentageOutput, and set the variables <atBats> and <hits> to zero:

```
CREATE FLEX Percentage IN AtBats, StatRequest
OUT OUTPUT STREAM PercentageOutput SCHEMA (atBatPercentage float)
BEGIN
   ON StatRequest {
      integer atBats := 0;
      integer hits := 0;
```

5. Close the Flex operator by dividing the number of <hits> by the number of <atBats> to calculate the batting average, <percentage>, placing that value in the <atBatPercentage> column of the output stream, and outputting it:

```
      float percentage := hits/atBats;
      output [atBatPercentage=percentage];
   };
END;
```

6. Between the initialization and close of the Flex operator, add a CCLScript routine to iterate through the entries in the Batting table to find out how many times the player was at bat and how many times he hit a ball. There are three ways to do this:
   a. Use the for and if commands to handle iterating through the records, and take advantage of the <AtBats_stream> that is automatically created with the **AtBats** reference query:

```
       for ( record in AtBats_stream ) {
          if ( record.player = StatRequest.player ) {
             if ( record.bases > 0 ) {
                hits := hits + 1;
             }
             atBats := atBats + 1;
          }
       }
```

   This method has the advantage of being easy to read and maintain.

   b. Use the modified for command syntax that incorporates testing whether the player name from the input stream matches the player name in the table entry:

```
       for ( record in AtBats_stream where player = StatRequest.player ) {
          if ( record.bases > 0 ) {
             hits:= hits + 1;
          }
          atBats := atBats + 1;
       }
```

   This method has the advantage of reducing processing time, because the records of other players are never pulled from the reference table.

c. Explicitly create and modify the iterator to use across the table, again taking advantage of the <AtBats_stream> that is automatically created with the AtBats reference query:

```
 AtBats_iterator := getIterator( AtBats_stream );
resetIterator( AtBats_iterator);
setRange( AtBats_iterator, player, StatRequest.player );
typeof(AtBats) result := getNext( AtBats_iterator );
while (not(isnull(result))) {
  if ( result.bases > 0 ) {
    hits := hits + 1;
  }
  atBats := atBats + 1;
  result := getNext( AtBats_iterator );
}
```

This method has the advantage of giving you the most explicit control over the processing.

6.8.6 Using a Reference Table Query in CCLScript to Get a Single Row from a Table

This example uses a reference table query within a CCLScript routine to obtain a single row from a table.

Context

In this example, we want to obtain a list of chain restaurants that have a franchise in a specified city.

Procedure

1. Obtain the necessary information about the external database table containing the row you want to obtain:
   - The name of the table containing the data
   - The schema for that table
   - The service to use to connect to it
2. Create a reference table query in your project.

   In this example, the reference table name is restaurants:

```
CREATE REFERENCE restaurants
SCHEMA (storeID integer, city string, name string, address string)
PROPERTIES service='HANA', source='restaurants';
```

   Because we have not specified a primary key in this reference, you can omit the sourceSchema property.
3. Create a second reference table query in your project.

   In this example, the reference table query name is chainList:

```
CREATE REFERENCE chainList
SCHEMA (chainID integer, name string, headquarterAddress string)
```

```
PROPERTIES service='HANA', source='chainList';
```

4. Create the input stream for events to enter the project.

   In this example, the input stream is named `restaurantQuery`, and consists of a single string, `city`:

   ```
   CREATE INPUT STREAM restaurantQuery SCHEMA (city string);
   ```

5. Initialize the Flex operator, named `optionQuery`, to read in the reference table queries `restaurants` and `chainLis`, and `restaurantQuery`. Output the restaurant chain names in the output stream `chainsInCity` when it receives the name of a city in the `restaurantQuery` stream:

   ```
   CREATE FLEX optionQuery
   IN restaurants, chainList, restaurantQuery
   OUT OUTPUT STREAM chainsInCity
   SCHEMA (name string, address string)
   BEGIN
      ON restaurantQuery {
   ```

6. Add a CCLScript routine to produce the list of chains that have a presence in the city. For each restaurant chain in the `chainList` table, it grabs the first instance of that chain name in the `restaurants` table whose address is in the city specified in the `restaurantQuery` input stream and outputs the restaurant chain name and address.

   ```
         for (record in chainList_stream) {
            typeof(restaurants) inCity :=
   restaurants_stream{[city=restaurantQuery.city;name=record.name;]};
            output [name=inCity.name;address=restaurants.address;];
         }
   ```

7. Since you are not performing any calculations at this time, close the Flex operator.

   ```
      };
   END;
   ```

6.9 Creating and Modifying Compute Queries

In the visual editor, produce a simple query that transforms the schema or field values of each incoming record. Each incoming event produces one new output event from the fields defined by the column expressions.

Context

The compute query creates in-line expressions in a stream or window. In the visual editor, the compute query is a shape within the palette. In the CCL editor, the compute query shape becomes a SELECT statement with column expressions.

Procedure

1. Open the SAP HANA Streaming Development perspective.
2. In the visual editor palette, in **Streams and Windows**, select **Compute** ( ).
3. Select a location in the diagram and click to add the shape.
4. Attach the compute object to the stream or window that provides input to this query.

   Attach compute objects to any stream, window, or Flex operator. Compute objects can have only one input. Any attempt to connect more than one input source is blocked.
5. Add columns:
   a. Click **Copy Columns from Input** ( ) in the shape toolbar to copy input fields into the schema for this query.
   b. Add additional columns by clicking **Add Column Expression** in the shape toolbar.
   c. Edit a column expression by double-clicking to open the inline editor, by selecting one of the tabs in the Properties view, or by selecting an expression and pressing **Ctrl+F2** to edit it using the pop-up expression editor.

   > Tip
   >
   > When entering column names and their datatypes, use **Tab** to easily move between cells in the table.
6. Add column expressions , as necessary.
7. Modify column expressions by selecting and modifying them directly, or by editing the corresponding fields in the Properties view.
8. Use the toggle option to designate the compute object as INPUT or OUTPUT. By default, computes are OUTPUT.

> Example
>
> You can also create compute queries in CCL, by using in-line expressions:

```
CREATE OUTPUT WINDOW Compute2 PRIMARY KEY DEDUCED KEEP 10 ROWS
AS
    SELECT
        InputWindow1.Column1 Column1 ,
        InputWindow1.Column2 + InputWindow1.Column3 theSum FROM InputWindow1 ;
```

6.10 Connecting a Stream to a Derived Window

Use a GROUP BY clause or the `nextval()` function to connect a stream to a derived window as part of a complex query.

A derived window is a stateful element that requires either an explicit or deduced primary key. When connecting a stream to a derived window, you assign a primary key by using either a GROUP BY clause or the `nextval()` function. Use the GROUP BY clause to aggregate column expressions from the stream to deduce a primary key when you compile the project in studio. You cannot explicitly specify a primary key using the GROUP BY clause. Use the `nextval()` function to assign an explicit primary key in the absence of a GROUP BY clause.

When using the GROUP BY clause, you should only use additive functions, such as `valueInserted()`, `sum()`, `avg()`, and `count()`, for all column expressions other than the GROUP BY columns. Additive functions allow the server to optimize the aggregation so that additional aggregation indexes are not maintained, which avoids unwanted memory consumption, as an aggregation index holds all of the records in memory. This improves the performance of the aggregation operation considerably. For more information, see Improving Aggregation Performance [page 343]. If using only additive functions is not an option, specify a retention policy other than KEEP ALL on the stream to limit the growth of the aggregation index.

In this section:

6.10.1 Connecting a Stream to a Derived Window Using the GROUP BY Clause

Use the GROUP BY clause to set a primary key for a derived window that is connected to a stream.

Procedure

1. Open a new or existing smart data streaming project in studio.
2. Connect a stream to a derived window.
3. In the derived window, click **Add Edit Group by Clause** ({ }).
4. Add the number of columns to group together. Click **OK**.

6.10.2 Connecting a Stream to a Derived Window Using nextval()

Use the `nextval()` function to set a primary key for a derived window that is connected to a stream.

Procedure

1. Open a new or existing smart data streaming project in studio.
2. Connect a stream to a derived window.
3. In the derived window under column expressions, right-click the wildcard and select **Delete Element**.
4. Select **Copy Columns from Input** to add the column expressions from the input window.
5. Click the dropdown menu for the derived window and select **Modify** > **Edit Primary Keys**.
6. In the dialog box, add a column as the primary key. Click **OK**.
7. Right-click the expression value for the primary key column and select **Edit Expression Value**.
8. Delete the expression value and replace it with `nextval()`. Click **OK**.

7 Exposing a Streaming Project as a Web Service

To expose a project as a Web service using one of the SAP HANA smart data streaming Web Services providers, set the Web Service Enabled option to true in the streaming project configuration (`.ccr`) file.

Prerequisites

- Start the Web Services Provider on which you are exposing the project.
- To run a project in a workspace other than the default, ensure that one or more connected workspaces are available.

> **Note**
>
> If you are using the Streaming Web Service, you do not need to set the Web Service Enabled option.

Procedure

1. Select **File** > **Open** > **Project...** and open the project you want to run.
2. Double-click the project configuration file (`<project-name>.ccr`) to open the CCR Project Configuration editor.
3. Select the **Advanced** tab.
4. Choose a project deployment type from the Project Deployment Details window. The following table describes the project deployment type options:

Type	Description
Non-HA	Non-HA deployments create one project option item and one instance item as children under the project deployment item.
HA	HA deployments create one project option item and two instance items as children under the project deployment item. HA provides for hot project failover between instances.

5. Set the value of the Web Service Enabled option to **true** in the Project Deployment Details window.
6. Save the updated `ccr` file.
7. For the changes to take effect, choose one of the following:
   - Use studio or `streamingclusteradmin` to stop and remove the project from the node, then redeploy (add) the project.

- Restart the cluster on which the project runs.

8. To run the project, choose one of the following:
    - Click **Run Streaming Project** in the main toolbar (in either the SAP HANA Streaming Development or the SAP HANA Streaming Run-Test perspective) to run the project in the default workspace.
    - Click the dropdown arrow next to the Run Project tool and choose **Run Streaming Project in Workspace**. Then select the workspace where this project will run.

    The project runs and shows results in the SAP HANA Streaming Run-Test perspective.

8 Zero Data Loss

Zero data loss protects a project against data loss in the event of a client failure, server failure, or both. Achieving zero data loss requires a judicious use of log stores when you set up the project, as well as configuration of project options and clients that use guaranteed delivery (GD).

With zero data loss:

- SAP HANA smart data streaming recovers streams or windows protected by one or more log stores to a consistent state as of the most recent checkpoint. (Any uncheckpointed data is lost and must be sent again by the publisher.)
- You can be confident you will not miss any events.
- Clients can minimize the number of duplicate events they receive by controlling how frequently they issue GD commits.
- Publishers can ensure that the data they publish is fully processed by the server and thereby reduce transmission of duplicates when the server restarts.
- You can optionally configure the server to control how frequently it issues automatic checkpoints and thus control how much uncheckpointed data is liable to be lost on a server failure.
- At the expense of performance, you can minimize (but not fully eliminate) the production of duplicate rows on server or subscriber restart by tweaking how frequently the server checkpoints data and how frequently GD subscribers issue GD commits.

In this section:

Guaranteed Delivery [page 214]

Guaranteed delivery (GD) is a delivery mechanism that supports high availability. It ensures that data continues to be processed from a stream or window even if the streaming server fails, the destination (third-party server) fails, or the destination does not respond for a period of time.

Consistent Recovery [page 219]

Consistent recovery lets you set up a project that can recover its data if it is interrupted by a server or connection failure. This feature can restore all the streams and windows in a project to a consistent state after a server or connection failure.

Auto Checkpoint [page 220]

Zero data loss relies on data being checkpointed (that is registered and saved in project log stores). Auto checkpoint lets you configure the checkpoint interval—the number of input transactions that triggers a checkpoint.

Achieving Zero Data Loss [page 221]

A lost connection or a server crash can cause data produced by a project to be lost before it is delivered to a listening client. If you cannot afford to lose data, complete the following tasks to configure zero data loss:

8.1 Guaranteed Delivery

Guaranteed delivery (GD) is a delivery mechanism that supports high availability. It ensures that data continues to be processed from a stream or window even if the streaming server fails, the destination (third-party server) fails, or the destination does not respond for a period of time.

Guaranteed Delivery

Guaranteed delivery (GD) uses log stores to ensure that a GD subscriber registered with a GD stream or window receives all the data processed by that stream or window even if the client is not connected when the data is produced. GD is supported on streams and windows and each GD stream or window requires a log store.

You can specify GD support for a window in the studio visual editor, or in CCL. A GD window supports multiple GD subscribers as well as both GD and non-GD subscriptions.

To use GD:

- Assign a log store to every stream and window in the project that cannot be recovered by an upstream provider.
- Do at least one of the following:
  - Enable GD on any bindings for the project.
  - Enable GD on project adapters that support it.
  - Use the C++ SDK, the Java SDK, or the .NET SDK to configure publishers sending data to your project to retransmit any data for which they do not receive a commit confirmation.
  - Use the C++ SDK, the Java SDK, or the .NET SDK to set up GD subscriptions for client applications. For more information, see the instructions on subscribing with guaranteed delivery in the *SAP HANA Smart Data Streaming: SDK Guide*.

You can subscribe to streams and windows configured for GD using adapters, bindings, the SDKs, or the subscribe tool.

Adapters	Enable guaranteed delivery on the adapters that support it and configure the GD adapter parameters. See the *SAP HANA Smart Data Streaming: Adapters Guide* for information on adapter support for GD.
Bindings	Enable GD on any project bindings to ensure that data is delivered to remote projects. See Bindings between CCL Projects [page 33].
SDKs	In the SDKs, set up GD subscriptions so each client can receive data and checkpoint messages from your project. The client, in turn, periodically responds to the project server with a commit message reporting the sequence number of the latest event the client has processed. The server need not save events once it knows that all GD clients have processed them, so it can free up their space for other uses. See the *SAP HANA Smart Data Streaming: SDK Guide* for more information.
Subscribe Tool	For testing or simple use cases, use the `streamingsubscribe` tool to subscribe in GD mode. See the *SAP HANA Smart Data Streaming: Utilities Guide* for details.

Recommendations for Guaranteed Delivery Subscribers

Follow these recommendations to reduce your chances of receiving duplicate rows or inconsistent data after a subscriber or server restart.

- Make sure the project you subscribe to is running in consistent recovery mode. See Consistent Recovery [page 219], or topics in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for details on setting project deployment options for consistent recovery.
- Subscribe to a stream or window on which GD is enabled. You can identify GD-enabled streams and windows by using:
  - The `supports_gd` command in the `streamingprojectclient` utility. See the *SAP HANA Smart Data Streaming: Utilities Guide* for more information.
  - Commands in the SDKs. Ssee the *SAP HANA Smart Data Streaming: SDK Guide* for more information.
- Send data on to your client as it arrives or buffer it locally. Issue commits for only those messages for which you have received a checkpoint notification. If the client does not support commits, deliver only those messages that have been checkpointed by the server and cache the rest locally. This ensures that the client is always consistent with the server on a restart after a client or server failure.
- To minimize data loss, configure either:
  - The publisher to issue commits frequently. See the *SAP HANA Smart Data Streaming: SDK Guide* for more information.
  - The Auto Checkpoint project deployment option to control how frequently the client receives checkpoint messages. See Auto Checkpoint [page 220] or the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for details on setting project deployment options for consistent recovery.
- When the smart data streaming server sends a checkpoint message, send a commit to the client or send the buffered rows followed by a commit.
- Issue a GD commit with the last committed sequence number to ensure that the server does not send the data again the next time the server or the subscription restarts. However, if the server does not shut down cleanly, it resends committed events that were not checkpointed.

Persistent Subscribe Pattern

In some situations, you can also use a persistent subscribe pattern (PSP), which is an older delivery mechanism for high availability.

If possible, use guaranteed delivery rather than persistent subscribe pattern. GD uses CPU and memory resources more efficiently and is more flexible from a development standpoint, because it does not force you to decide how many subscribers will be supported when you set it up. However, you might prefer PSP if you do not want the guaranteed delivery store to be a log store for performance reasons. Using a memory store allows recovery when the client restarts, but not when the project restarts.

For more information, see Adding a Persistent Subscribe Pattern [page 217].

In this section:

Adding a Persistent Subscribe Pattern [page 217]
Set up a persistent subscribe pattern (PSP) for any element in a project.

Parent topic: Zero Data Loss [page 213]

Related Information

Consistent Recovery [page 219]
Auto Checkpoint [page 220]
Achieving Zero Data Loss [page 221]
SAP HANA Smart Data Streaming: CCL Reference
SAP HANA Smart Data Streaming: Configuration and Administration Guide
SAP HANA Smart Data Streaming: SDK Guide
SAP HANA Smart Data Streaming: Utilities Guide
SAP HANA Smart Data Streaming: Adapters Guide

8.1.1 Adding Guaranteed Delivery

Configure guaranteed delivery (GD) in studio for a stream or window to ensure that its output is delivered to subscribers.

Prerequisites

- Create at least one log store in which your guaranteed delivery stream or window can store its events and GD logs.
- In the CCR Project Configuration editor,
  - Make sure the Consistent Recovery project deployment option is enabled. (It is enabled by default in new projects.)
  - (Optional) Enable the Auto Checkpoint project deployment option.

Context

Set up a GD stream or window using a CREATE STREAM, CREATE WINDOW, or CREATE FLEX statement. Make sure to assign a GD log store to each GD-enabled stream or window. Do not create GD-enabled streams or windows inside modules—this is not supported because you cannot attach adapters or subscribe directly to elements in modules.

A stream or window configured for GD also supports non-GD subscriptions. Enabling GD does not significantly affect the stream or window’s performance when it has no registered GD subscribers.

You can subscribe to streams and windows configured for GD using adapters, bindings, the SDKs, or the subscribe tool.

Procedure

1. Choose the stream or window to support GD.
2. (Optional) If a window that you want to support GD does not exist, create a new window:
   a. In the visual editor palette, in Streams and Windows, select an option to create a stream or window.
      - Streams that support GD include **Input Stream**, **Derived Stream**, and **Flex**.
      - Windows that support GD include **Input Window**, **Derived Window**, **Flex**, **Aggregate**, **Compute**, and **Join**.
   b. Select a location in the diagram and click to add the stream or window.
3. Select the new element and look at its Properties view. (If the Properties view is not visible, from the main menu select **Window** > **Show View** > **Properties**.)
4. In the the Properties view:
   a. Select the **Properties** tab and click to check the **Supports Guaranteed Delivery** box.
   b. From the **Guaranteed Delivery Store** dropdown, select the log store you created for this element's GD logs.
   c. (For GD windows only) Select the **General** tab. In the **Store** field, enter the name of the log store you created for the GD window.

8.1.2 Adding a Persistent Subscribe Pattern

Set up a persistent subscribe pattern (PSP) for any element in a project.

Context

> **Note**
>
> Whenever possible, use guaranteed delivery instead of PSP. See Guaranteed Delivery [page 214] for more information.

Set up PSP for each subscriber individually. Enabling PSP on a data stream or window creates two new elements:

- A log window (a Flex operator with a log store)
- A truncate window (also called the control window)

The data stream or window on which PSP is enabled and the truncate window plug into the log window. The PSP-enabled stream or window delivers data to the log window. The log window generates a sequence number, takes the opcode from the data, and places them at the beginning of each row of data. The log window sends this data to the output adapter that is attached to it. The adapter, taking advantage of guaranteed processing features on the subscribing system (where the data will be consumed), determines when to notify the truncate window that rows marked for deletion have been processed by the subscriber. When it receives this notification, the truncate window informs the log window that the data that has been processed and the log window removes the data from its log store.

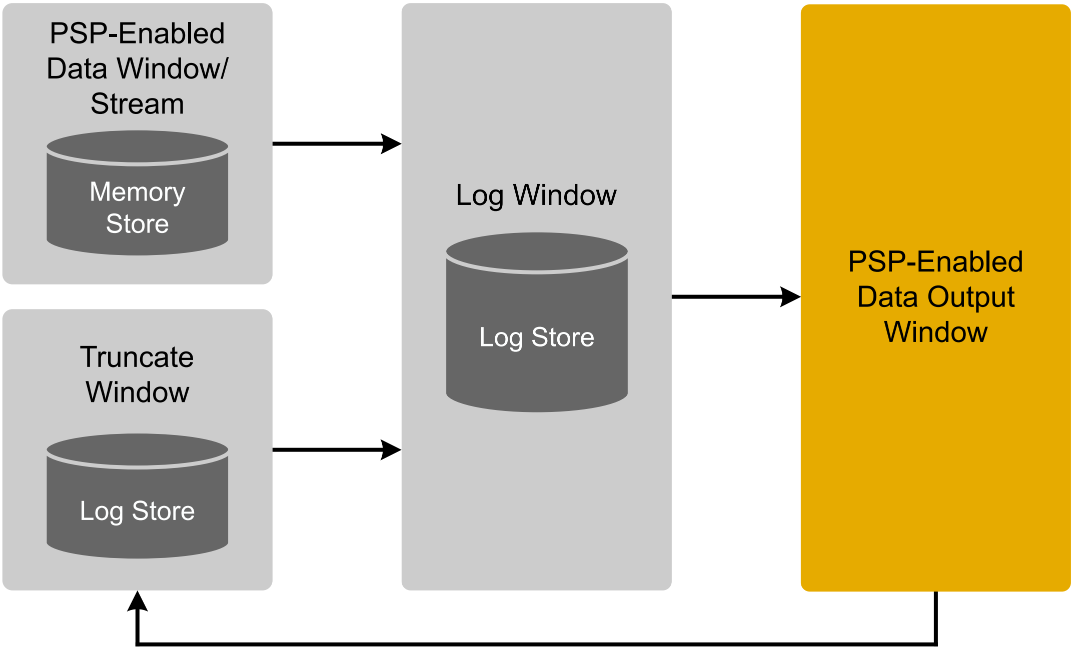


Figure 5: Per-Subscriber PSP Overview

Input adapters support persistent subscribe pattern (PSP) using facilities provided by the data source. Output adapters use PSP directly.

> **Note**
>
> The WebSphereMQ Input and Output adapters, all JMS Input and Output adapters, and the TIBCO Rendezvous adapter all support PSP. These adapters have specific, unique PSP and GD parameters. For more information, see the *SAP HANA Smart Data Streaming: Adapters Guide*.

Procedure

1. Open the project in the visual editor and choose the element to support PSP.
2. (Optional) If an element suitable for PSP does not exist, create a new one:
   a. In the visual editor palette, click a shape tool such as Streams and Windows.
   b. Click the shape (element) you want to create (**Input Stream**, for example).
   c. Select a location in the diagram and click to add the shape.
3. Right-click the shape and select **Modify** > **Add Persistent Subscribe Pattern**.
4. In the Select Store dialog, select either **Create New Store** or an existing store, then click **OK**.

   Studio creates two elements and attaches them to the shape you are configuring for PSP:

- A Flex operator named `<-PSP-shape-name>_log<number>`.
- A truncate window named `<-PSP-shape-name>_truncate<number>`.

The new truncate window and the shape you are configuring for PSP connect automatically to the new Flex operator.

Next Steps

- To complete the PSP set-up, attach an appropriate output adapter to the Flex operator.
- Repeat the steps above to add more subscribers.

8.2 Consistent Recovery

Consistent recovery lets you set up a project that can recover its data if it is interrupted by a server or connection failure. This feature can restore all the streams and windows in a project to a consistent state after a server or connection failure.

Recovery consistency depends on guidelines for log stores:

- When consistent recovery is enabled, the server uses coordinated checkpoints to save data in log stores.
- When any log store fails to complete a checkpoint, all the log stores for that project roll back to their state as of the previous successful checkpoint. This rule ensures that even if a server or connection fails, all log stores in a project are consistent with one another. However, any input data that has not been checkpointed is not recovered upon restart.

Enabling consistent recovery has no effect if there are no log stores in the project. When you enable consistent recovery for a project, place the log stores on a shared drive where all the machines in the smart data streaming cluster can access them.

Enable consistent recovery in the project configuration (CCR file), either in studio or by manually editing the CCR file. See *Project Configuration Files* in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

In consistent recovery mode, a project treats commits issued by publishers as checkpoint requests. When the publisher receives the return of a commit from the project, it can notify its source that the data in question has been processed.

All guaranteed delivery subscribers to a stream or window stored in a log store receive checkpoint notifications. GD subscribers can use this notification as an indication that it is safe to commit data in its target.

Consistent recovery works well with projects configured for cold failover if log stores are set up following the log store guidelines. When a project set for cold failover stops responding, the cluster restarts the project, typically on another host. Consistent recovery enables the restarted project to come back up to a consistent state corresponding to the last checkpoint.

> **Note**
>
> Avoid using consistent recovery mode in projects where active-active HA mode is also enabled. Because smart data streaming is nondeterministic, enabling consistent recovery mode on the instances of an active-

active project cannot be guaranteed to produce the same data in the primary and secondary instances if there is a failure.

When consistent recovery is not enabled (which is the default state), the project does not ensure that all the log stores recover to the same point in time after a server failure. Some log stores may recover to a checkpoint state earlier in time than other log stores because the checkpoints across log stores are not treated as an atomic operation. When there is only one log store in the project, this is not an issue.

When you use consistent recovery, the recommendation that all input windows in a project and their direct or indirect dependents be placed in the same log store no longer applies. Instead, consider using multiple log stores placed on different disks to improve performance. Using multiple log stores is possible because consistent recovery ensures that all the log stores in the project are always consistent with each other.

Parent topic: Zero Data Loss [page 213]

Related Information

Guaranteed Delivery [page 214]
Auto Checkpoint [page 220]
Achieving Zero Data Loss [page 221]
SAP HANA Smart Data Streaming: Configuration and Administration Guide

8.3 Auto Checkpoint

Zero data loss relies on data being checkpointed (that is registered and saved in project log stores). Auto checkpoint lets you configure the checkpoint interval—the number of input transactions that triggers a checkpoint.

Checkpoints are triggered when:

- A publisher in a client application issues a commit (if the consistent recovery project option is enabled)
- The project server determines a checkpoint is required
- The project processes the number of transactions you specified in the auto checkpoint option
- The project shuts down cleanly
- The project restarts after an unexpected shutdown

Auto checkpoint lets you control how often log store checkpoints occur across all input streams and windows in the project. More frequent checkpoints mean less data is lost if the server crashes. At the maximum checkpoint frequency of every input transaction (value of 1), all input data is protected except the data from the last transaction, which might not be checkpointed before a crash. Here, input transaction refers to one record, one transaction, or one envelope published to the server and currently in flight. This applies to a single publisher; if there are multiple publishers, then the last transaction for every publisher that is in flight may be lost. When you set checkpoint frequency, you make a trade-off: with frequent checkpoints you can reduce the amount of data at risk, but performance and latency may suffer as a result. The alternative is to increase performance but risk a larger amount of data loss by setting infrequent checkpoints.

Setting auto checkpoint guarantees that a checkpoint occurs at least every `<N>` rows, where `<N>` is the checkpoint interval. The checkpoint itself may include more input rows because the system ensures that all inputs (other than the input stream that triggered the checkpoint) have consumed all the data in its input queues. The actual checkpoint may happen earlier than called for by the auto checkpoint interval if the system decides it is necessary.

When the server completes a checkpoint, it sends checkpoint messages to GD subscribers to notify them that all data up to the sequence number specified in the checkpoint message can be safely recovered by the server on restart.

Setting auto checkpoint has no effect if there are no log stores in the project. Auto checkpoint is not dependent on consistent recovery; use it with consistent recovery enabled or disabled.

> **Note**
>
> You should only perform one of the following:
>
> - Enable auto checkpoint
> - Configure publishers sending data to the project to issue commits, which trigger checkpoints.

Parent topic: Zero Data Loss [page 213]

Related Information

Guaranteed Delivery [page 214]
Consistent Recovery [page 219]
Achieving Zero Data Loss [page 221]

8.4 Achieving Zero Data Loss

A lost connection or a server crash can cause data produced by a project to be lost before it is delivered to a listening client. If you cannot afford to lose data, complete the following tasks to configure zero data loss:

Task	For Instructions, see...
Create a project with one or more guaranteed delivery streams or windows or add a GD stream or window to an existing project. You can do this in studio or by adding CREATE STREAM, CREATE WINDOW, or CREATE FLEX statements to the project CCL file.	Adding Guaranteed Delivery [page 216] *SAP HANA Smart Data Streaming: CCL Reference*
Set up log stores for any streams or windows in the project that cannot be recovered by an upstream provider. Review the guidelines, restrictions, and sizing instructions for log stores to ensure that your project can accurately and completely recreate its data after a restart.	Creating a Log Store [page 160] and all subsections

<table>
<tr><th>Task</th><th>For Instructions, see...</th></tr>
<tr><td>Variables and CCLScript data structures (dictionaries, vectors, and event caches) do not persist in log stores and cannot be recovered after a failure. Use these structures with log stores only when:
You can provide logic to reconstruct the structures on restart.
Processing is not affected if the structures are missing after a restart.</td><td>---</td></tr>
<tr><td>Enable guaranteed delivery on any bindings associated with GD-enabled streams or windows.</td><td>Adding Guaranteed Delivery [page 216]</td></tr>
<tr><td>Before you deploy the project, enable these options in the project configuration (CCR) file:
Failover
(Optional) Auto Checkpoint

Note
You should enable Auto Checkpoint only if you do not configure publishers of inbound data to trigger checkpoints by issuing commits.
Consistent Recovery

Note
You should not enable active-active HA mode for projects where consistent recovery is enabled. Active-active mode does not safeguard against data loss or data inconsistency when the project switches from the primary instance to the secondary.</td><td>High Availability, as well as the respective option topics, in the SAP HANA Smart Data Streaming: Configuration and Administration Guide.</td></tr>
<tr><td>Enable guaranteed delivery on the project's adapters.

Note
Some adapters do not support guaranteed delivery. See the Adapter Summary in the SAP HANA Smart Data Streaming: Adapters Guide for information on GD support.</td><td>SAP HANA Smart Data Streaming: Adapters Guide for GD-specific adapter properties.</td></tr>
<tr><td>If auto checkpoint is not enabled, configure publishers sending data to your project to issue commits to trigger server checkpoints. When the commit call returns, the publisher knows that the server has checkpointed the data.

Note
When a project receives data from more than one publisher, a commit issued by one publisher triggers checkpointing of data sent by all publishers.

Configure publishers to retransmit any data for which they do not receive a commit confirmation.</td><td>SAP HANA Smart Data Streaming: SDK Guide</td></tr>
</table>

Task	For Instructions, see...
Set up guaranteed delivery subscriptions for client applications that consume data from your project. To confirm that subscribers have received and processed data, configure them to send GD commit calls in response to checkpoint notifications.	*SAP HANA Smart Data Streaming: SDK Guide*

Parent topic: Zero Data Loss [page 213]

Related Information

Guaranteed Delivery [page 214]
Consistent Recovery [page 219]
Auto Checkpoint [page 220]
SAP HANA Smart Data Streaming: CCL Reference
SAP HANA Smart Data Streaming: Configuration and Administration Guide
SAP HANA Smart Data Streaming: SDK Guide
SAP HANA Smart Data Streaming: Adapters Guide

9 Machine Learning with Streaming

Predictive analysis defines functions that can perform analytic algorithms. Incremental machine learning algorithms learn and update a model on the fly, so that predictions are based on a dynamic model.

Traditional supervised learning algorithms train data models based on historical, static data. In the traditional scenario, training and retraining are infrequent events that require a large pre-existing data set to be maintained. Once training is complete, a learned model is stored in a table. When new data arrives, the scoring function makes predictions based on this stored model. As patterns change, the model needs to be retrained with more historical, labelled, data to ensure the accuracy of the algorithm. In contrast, supervised learning in streaming can continuously learn as new data arrives and is labelled, thus allowing accurate scoring in real time, which adapts to changing situations.

Traditional unsupervised learning algorithms analyze a large dataset to detect hidden patterns in data, without any labels being provided to the algorithm. When new data needs to be analyzed, the entire dataset must be re-examined to determine patterns. Conversely, unsupervised learning in streaming is able to detect novel patterns in streaming data in real time without any re-analysis of previously examined data.

By combining smart data streaming with integrated machine learning algorithms, you can leverage both supervised and unsupervised learning to train models, score and cluster data all in real time with modest memory and storage requirements.

You can use these functions from within a streaming project using CCL elements. To use machine learning functions in a streaming project, you need to define a predictive analysis model, and configure machine learning function parameters. You can add, edit or delete models within SAP HANA studio, using the data services view in the SAP HANA Streaming Development perspective. Once they are configured, you can use saved models in streaming projects to run analytic algorithms on sets of incoming data.

In this section:

9.1 Workflow: Using Machine Learning with Streaming

Create models that use machine learning functions, insert them into streaming projects, and set up the projects to apply machine learning functions to streaming data.

Procedure

1. In SAP HANA studio, open the SAP HANA Streaming Run-Test perspective, and connect to the streaming server. In the server view, either:
   - Select **Add New Server URL** and configure a new streaming server connection, or;
   - Right-click on an existing server connection and select **Connect Server**.
2. In the SAP HANA Streaming Development perspective, create or load an existing SAP HANA service through the data services view. See *Adding a Connection to an SAP HANA Database* in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

   > Note
   >
   > The permissions enabled on the data service also apply to all models using this data service.

3. Open the model folder of your data service:
   a. Right-click on the server and select **Load Workspaces**.
   b. Right-click on the workspace and select **Load Services**.
   c. Right-click on the data service and select **Discover**, then right-click on the **Models** folder and select **Discover** again.
4. Right-click on the **Models** folder and select **Add Model**, or select an existing model to update. Fill out the required properties and parameters. See Model Properties and Parameters Reference [page 234].

   Every kind of machine learning model requires its own specific input and output schema format. See the individual function topics for more information:
   - Hoeffding Tree Training for Classification [page 227]
   - Hoeffding Tree Scoring for Classification [page 229]
   - DenStream Clustering [page 230]
5. Open an existing streaming project, or create a new project. See Creating or Modify a Streaming Project in SAP HANA Studio [page 71].
6. Create an input stream using the same schema as your selected model.

   > Note
   >
   > Column names are case-sensitive. Make sure that the schema of the input stream matches the schema in the model. If the schemas do not match, the model stream cannot read the data, and interprets events as null. See *Appendix: Troubleshooting>Model Reads Rows as Null*.

7. Add the model to the project using one of the following methods:
   - In the visual editor, add a **Model** element from the palette.

- In the CCL editor, use a DECLARE statement in the following format:

```
DECLARE MODEL <model-name>
TYPE MACHINE_LEARNING
INPUT SCHEMA (<input-schema>)
OUTPUT SCHEMA(<output-schema>)
PROPERTIES
    Dataservice = '<service-name>' ;
```

8. Add a model stream to the project using one of the following methods:
    - In the visual editor, add a **Model Stream** element from the palette.
    - In the CCL editor, use a CREATE STREAM statement in the following format:

```
CREATE STREAM <model-stream-name>
AS
    EXECUTE MODEL <model-name>
    FROM <input-stream-name>;
```

9. In the visual editor, attach the model and input stream elements to the model stream.
10. (Optional) Create a derived stream to gather the results of the data analysis, and redirect it to the appropriate SAP HANA table.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide

9.2 Model Management

Models help set the required variables for your machine learning function, and can be reused in multiple projects.

Smart data streaming supports three kinds of predictive analytics incremental machine learning functions:

- **Hoeffding Tree Training for Classification**, which continuously works to discover predictive relationships, even as the streaming data changes.
- **Hoeffding Tree Scoring for Classification**, which applies the predictive model to the data.
- **DenStream Clustering**, which groups and prunes data object points, based on their weighted significance.

Combining the training and scoring classification algorithms, supervised learning in smart data streaming can continuously learn as new data arrives and is labelled, thus allowing accurate scoring in real-time, which adapts to changing situations. Using the clustering algorithm, unsupervised learning in smart data streaming is able to detect novel patterns in streaming data in real-time without any re-analysis of previously examined data.

Every kind of machine learning model requires its own specific input and output schema format. See Hoeffding Tree Training for Classification [page 227], Hoeffding Tree Scoring for Classification [page 229], and DenStream Clustering [page 230].

You can create models in the SAP HANA Streaming Development perspective visual editor, or you can introduce a model into CCL using the DECLARE MODEL construct:

```
DECLARE MODEL ModelName
TYPE MACHINE_LEARNING
INPUT SCHEMA (id integer, col1 string, col2 string)
OUTPUT SCHEMA (id integer, col1 string, col2 string)
PROPERTIES
  dataservice = 'service1';
```

To declare a model, you need a preconfigured data service. Your model uses this data service to connect to the SAP HANA database to store and retrieve data.

In this section:

9.2.1 Hoeffding Tree Training for Classification

The training model is based on a classification algorithm, which uses streaming decision tree induction derived from the Hoeffding bound.

The Hoeffding bound says that with a probability of 1- δ , the mean of a random variable of range R won't differ from its estimate after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}}$$

The Hoeffding bound can give consistent results regardless of the probability distribution generating observations (the number of observations required to reach certain values may differ across distributions). The algorithm also detects concept drift, and updates models accordingly. See the Model Properties and Parameters Reference [page 234] for more information about each property and parameter.

The degree to which you train before beginning to score may depend on a number of factors. However, at a minimum, scoring should not begin before the first sync point. See Hoeffding Tree Scoring for Classification [page 229].

Input and Output Schemas

The input schema for the training algorithm must be in the format `[IDS]+S`, where:

- `[IDS]+` specifies the data features. Each feature column can be either an `integer`, a `double`, or a `string`. There is no limit to the number of feature columns.
- `S` specifies the label. This column must be a `string`.

The output schema for the training algorithm must be in the format `D`, which displays the accuracy of the training model. The value is a number between 0 and 1, representing the percentage of accuracy. This column must be a `double`.

The training model has a limit for memory use, works in a limited amount of time, and is constantly ready to predict. You can create a model in the SAP HANA Streaming Development perspective. See Creating a Model [page 232].

Usage

To train a model, first set up the model properties and parameters, then use the following basic CCL structure for your streaming project:

```
DECLARE MODEL <model-name>
TYPE MACHINE_LEARNING
    INPUT SCHEMA (<input-schema>)
    OUTPUT SCHEMA (<output-schema>)
    PROPERTIES
        dataservice = '<data-service-name>' ;
CREATE INPUT STREAM <input-stream-name>
    SCHEMA (<stream-schema>) ;
CREATE OUTPUT STREAM <model-stream-name>
     AS EXECUTE MODEL <model-name>
    FROM <input-stream-name> ;
CREATE OUTPUT STREAM <output-stream-name>
    AS SELECT *
```

```
    FROM <model-stream-name> ;
```

> **Note**
>
> Make sure the model name is unique in the model metadata table. At run time, there can be only one training stream for a given model.

See Example: Hoeffding Tree Training and Scoring [page 238] for a use-case example.

9.2.2 Hoeffding Tree Scoring for Classification

Scoring functions make calculations based on training data.

A scoring model must reference a training model. Without trained data, scoring cannot occur. At run time, the scoring stream periodically checks whether the training model content is updated, and grabs the latest training information when it appears.

The sync point property determines when your scoring function syncs with the database to pull in trained data. At minimum, the scoring function needs to sync with the database at the same time as, or slightly after the first training sync point. For example, if the training function syncs with the database every 1000 events, the scoring sync point should be 1000 events or higher. See the Model Properties and Parameters Reference [page 234] for more information about each property and parameter.

> **Note**
>
> While you can start scoring as soon as there is some trained data in the database, the scoring accuracy increases in proportion to the training accuracy. For the best results, wait until training accuracy reaches your desired percentage, for example, 0.95 (95%).

Input and Ouput Schemas

The input schema for the scoring algorithm must be in the format `[IS][IDS]+`, where:

- `[IS]` specifies the ID column. This column can be either an `integer` or a `string`.
- `[IDS]+` specify the data features, which correspond to the referenced training model. Each feature column can be either an `integer`, a `double`, or a `string`.

The output schema for the scoring algorithm must be in the format `[IS]SD`, where:

- `[IS]` specifies the ID column. This column can be either an `integer` or a `string`.
- `S` specifies the prediction class. This column must be a `string`.
- `D` specifies the probability that the prediction class will appear. This column must be a `double`.

You can create a model in the SAP HANA Streaming Development perspective. See Creating a Model [page 232].

Usage

To score a model, first set up the model properties and parameters, then use the following basic CCL structure for your streaming project:

```
MODEL <model-name>
TYPE MACHINE_LEARNING
    INPUT SCHEMA (<input-schema>)
    OUTPUT SCHEMA (<output-schema>)
    PROPERTIES
        dataservice = '<data-service-name>' ;
CREATE INPUT STREAM <input-stream-name>
    SCHEMA (<stream-schema>);
CREATE OUTPUT STREAM <model-stream-name>
    AS EXECUTE MODEL <model-name>
    FROM <input-stream-name> ;
CREATE OUTPUT STREAM <output-stream-name>
    AS SELECT *
    FROM <model-stream-name> ;
```

> **Note**
>
> Make sure the model name is unique in the model metadata table.

See Example: Hoeffding Tree Training and Scoring [page 238] for a use-case example.

9.2.3 DenStream Clustering

This model uses the DenStream algorithm to discover data object clusters of arbitrary shape, handle data outliers and noise, and by pruning, maintain only necessary information for data clustering in order to limit memory consumption.

The statistical properties of the incoming streaming data change over time, and in unforeseen ways. To remain accurate, the DenStream algorithm runs constant updates. Old data or clusters are less important, and outliers may evolve into clusters. We cannot keep all the data points from the stream, so instead we form core-micro-clusters by first identifying potential core-micro-clusters and outlier-micro-clusters. To accomplish this, the algorithm uses the following formula:

$$T_p = \left[\frac{1}{\lambda}\log\left(\frac{\beta\mu}{\beta\mu - 1}\right)\right]$$

For potential micro-clusters, if a data point merges with no other point, its weight will gradually decay. If the weight falls below beta mu, the point becomes an outlier, and is pruned and deleted. The weight of any new cluster is small compared with existing clusters, but we must consider that an outlier-micro-cluster may grow into a potential core-micro-cluster. The total number of micro-clusters increases logarithmically with increasing stream length, but we specify an upper bound of memory cost.

There are two steps for data clustering: an online phase of micro-cluster maintenance (performing a statistical summary of the input data), and an offline phase of generating the final clusters (based on the statistical summary). The clustering condition parameter specifies when the clustering gets triggered—either periodically, by a specific number of input events, or after a length of time. See the Model Properties and Parameters Reference [page 234] for more information about each property and parameter.

Input and Output Schemas

The input schema for the DenStream clustering algorithm must be in the format `[IS][IDS]+`, where:

- `[IS]` specifies the ID column. This column can be either an `integer` or a `string`.
- `[IDS]+` specify the data features, which correspond to the referenced training model. Each feature column can be either an `integer`, a `double`, or a `string`. There is no limit to the number of feature columns.

The output schema for the DenStream clustering algorithm must be in the format `[IS]ISDISS`, where:

- `[IS]` specifies the ID column. This column can be either an `integer` or a `string`.
- `I` specifies the cluster ID. This column must be an `integer`.
- `S` specifies the measure of evalution. This column must be a `string`.
- `D` specifies the evaluation value. This column must be a `double`.
- `I` displays the phase of the clustering algorithm, where 0 is initialization, 1 is the offline phase, and 2 is the online phase. This column must be an `integer`.
- `S` displays the clustering results. This column must be a `string`.
- `S` displays the outlier information. This column must be a `string`.

You can create a model in the SAP HANA Streaming Development perspective. See Creating a Model [page 232].

Usage

To use a clustering model, first set up the model properties and parameters, then use the following basic CCL structure:

```
DECLARE MODEL <model-name>
TYPE MACHINE_LEARNING
    INPUT SCHEMA (<input-schema>)
    OUTPUT SCHEMA (<output-schema> )
    PROPERTIES
        dataservice = '<data-service-name>' ;
CREATE INPUT STREAM <input-stream-name>
    SCHEMA (<stream-schema>) ;
CREATE OUTPUT STREAM <model-stream-name>
    AS EXECUTE MODEL <model-name>
    FROM <input-stream-name> ;
CREATE OUTPUT STREAM <output-stream-name>
    AS SELECT *
    FROM <model-stream-name> ;
```

> **Note**
>
> Make sure the model name is unique in the model metadata table. At run time, there can be only one clustering stream for a given model.

See Example: DenStream Clustering [page 240] for a use-case example.

9.2.4 Creating a Model

You can create reusable models for applying machine earning functions.

Prerequisites

Create and connect to a data service. See *Configuring External Database Access* in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

Procedure

1. In the data services view of the SAP HANA Streaming Development perspective, right-click on a data service and select **Discover**.
2. Right-click on the **Models** folder, and select **Add Model**.
3. In the studio properties view, fill in the model properties and parameters, if applicable.

   See the Model Properties and Parameters Reference [page 234] for more information about each property and parameter.

   > **Note**
   >
   > The Hoeffding Tree scoring function requires an existing Hoeffding Tree training model. Before defining a scoring model, create a training model using the same data service.
4. Right-click on the model you created, and select **Save Model**.
5. (Optional) To set a custom name, right-click on the new model and select **Rename Model**.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide

9.2.5 Updating a Model

You can update a model definition through the SAP HANA Streaming Development perspective data services view.

Context

If you update any model parameters, and then restart a project using the same model, the model overwrites any previously stored data with new content, based on the updated parameters. You can leverage this behavior to restart the training or clustering process (since updating the parameters implies that the training or the clustering process will re-initiate).

Procedure

1. In the data services view of the SAP HANA Streaming Development perspective, right-click on a data service and select **Discover**.
2. Right-click on the **Models** folder and select **Discover**.
3. Select a model and edit its properties and parameters in the properties view.
4. Right-click on the model in the data services view and select **Save Model**.

9.2.6 Clearing or Deleting a Model

You can clear or delete a model definition through the SAP HANA Streaming Development perspective data services view.

Context

There are two methods to completely clear out model content and start from scratch: clearing a model, or deleting a model. Clearing a model only removes model content and evaluation data, with the model still remaining on the data service, and the model metadata remaining in the database tables. Deleting a model removes all model metadata from the database tables, as well as its content and evaluation data, and the model disappears from the data service.

> **i Note**
>
> If you clear a model while a project is running, the project keeps running. However, the machine learning function in the project begins learning from scratch, starting with the most recent event.

Procedure

1. In the data services view of the SAP HANA Streaming Development perspective, right-click on a data service and select **Discover**.
2. Right-click on the **Models** folder and select **Discover**.
3. Right-click on the model and select either **Clear Model** or **Delete Model**.

9.2.7 Model Properties and Parameters Reference

Using studio, fill out the applicable properties and parameters when setting up a machine learning model.

Properties

Fill out the following properties when setting up any machine learning model in smart data streaming.

Property	Description
Description	The description of the model.
Machine Learning Function	One of the available machine learning functions: Hoeffding Tree Training, Hoeffding Tree Scoring, or DenStream Clustering.
Input Schema	The schema of the input data. Requires a comma-delimited list of datatypes. The schema is a regular expression in the following format: • Training: `[IDS]+S` • Scoring: `[IS][IDS]+` • Custering: `[IS][IDS]+` where `I` is `integer`, `D` is `double`, and `S` is `string`. For an in-depth explanation of each function's input schema, see the topics Hoeffding Tree Training for Classification [page 227], Hoeffding Tree Scoring for Classification [page 229], and DenStream Clustering [page 230].
Output Schema	The desired schema for the output data. Requires a comma-delimited list of datatypes. The schema is a regular expression in the following format: • Training: `D` • Scoring: `[IS]SD` • Clustering: `[IS]ISDISS` where `I` is `integer`, `D` is `double`, and `S` is `string`. For an in-depth explanation of each function's output schema, see the topics Hoeffding Tree Training for Classification [page 227], Hoeffding Tree Scoring for Classification [page 229], and DenStream Clustering [page 230].

Property	Description
Memory Quota (MB)	The maximum amount of memory that smart data streaming can use for the model. If the model exceeds the memory quota, the application shuts down, and prompts you to either increase the quota, or clear the model content.
Sync Point	The interval, in seconds (S) or number of rows (N), that determines when smart data streaming backs up model content. If the project shuts down unexpectedly, smart data streaming uses the most recent backup to continue training, scoring, or clustering. Each function syncs with the SAP HANA database for a slightly different purpose: • Training models create backups on sync, which help recover model content after an unexpected shutdown, and retrain the model. • Scoring models create backups on sync, and gather the latest trained data. • Clustering models create backups on sync, which help recover model content after an unexpected shutdown.
Referenced Model	(Used only by the Hoeffding Tree scoring function) The training model to use for scoring algorithms.

Parameters

The following parameters are used solely by the Hoeffding Tree training function:

Parameter	Datatype	Default Value	Description
Split Criterion	`integer`	0	The information gain. The only supported value is 0.
Split Threshold	`integer`	50	The minimum number of records required in the tree node before the algorithm splits the node into new branches.
Max Bucket Size	`integer`	4	The maximum amount of data allowed in a container to detect concept drift.
Tau	`double`	0.05	The value used for tie-breaking. Tau is used when the difference between two attributes is very small, and the information gain is too similar to otherwise differentiate between the attributes.
Delta	`double`	0.01	The probability that one attribute is superior to others when the observed difference of information gain is greater than epsilon. For example, when delta is 0.05, the probability is 95%.

The following parameters are used solely by the DenStream clustering function:

Parameter	Datatype	Default Value	Description
INIT N	`integer`	1000	The initial number of events that trigger the clustering algorithm.

Parameter	Datatype	Default Value	Description
Epsilon	`double`	16	The size of the clustering neighborhood. Objects within this radius form into one object. A large epsilon may disrupt clustering.
Beta	`double`	0.2	The outlier threshold.
Mu	`double`	10	The average number of points in each cluster. A smaller mu implies a larger number of micro-clusters. **Note** The product of beta and mu must be greater than 1.
Lambda	`double`	0.25	The decay factor.
Clustering Condition	`long`	1000	The number of events that need to occur before clustering begins. Clustering reoccurs every time the events reach the specified number. The clustering algorithm gathers events in the offline phase, and when the number reaches the specified clustering condition, the algorithm enters the online phase and begins clustering.
Max. Number of Categories	`integer`	0	The maximum number of attributes in a column. Setting the value to zero allows the algorithm to prepare for any number of number of columns; however, this can impact performance. For example, if the column is labeled "gender", you can expect the values to be "male" or "female", and therefore set the maximum number of categories to 2. **Note** If the number of attributes exceeds the specified value during run time, the algorithm reallocates more memory.

9.2.8 Model Database Table Reference

Machine learning models for streaming data store information in SAP HANA database tables, under the user schema.

When you create and save a streaming model for the first time, studio creates three tables in the SAP HANA database. Consult these tables for overview information of all the models you have created. Studio makes relevant changes to the SAP HANA database tables whenever any changes occur to the models:

- If you update a model, studio also updates the model database tables to reflect the changes.
- If you clear a model, you clear the content table and evaluation table entries for that model, but not the model metadata. This way, the algorithm of the cleared model begins learning from the beginning.
- If you delete a model, you also delete all model information from the database tables, including the model metadata.

The STREAMING_ML_MODEL_METADATA_ table contains a snapshot of the model properties:

Property	Description
NAME	The name of the model.
DESCRIPTION	The model description.
PAL_FUNCTION	One of the available machine-learning functions: Hoeffding Tree Training, Hoeffding Tree Scoring, or DenStream Clustering.
INPUT_SCHEMA	The schema of the input data, presented as a comma-delimited list of datatypes. The schema is a regular expression in the following format: • Training: `[IDS]+S` • Scoring: `[IS][IDS]+` • Clustering: `[IS][IDS]+` where `I` is `integer`, `D` is `double`, and `S` is `string`. For an in-depth explanation of each function's input schema, see the following topics: • Hoeffding Tree Training for Classification [page 227] • Hoeffding Tree Scoring for Classification [page 229] • DenStream Clustering [page 230]
OUTPUT_SCHEMA	The schema of the output data, presented as a comma-delimited list of datatypes. The schema is a regular expression in the following format: • Training: `D` • Scoring: `[IS]SD` • Clustering: `[IS]ISDISS` where `I` is `integer`, `D` is `double`, and `S` is `string`. For an in-depth explanation of each function's input schema, see the following topics: • Hoeffding Tree Training for Classification [page 227] • Hoeffding Tree Scoring for Classification [page 229] • DenStream Clustering [page 230]
PARAMETERS	(Used by the Hoeffding Tree training and DenStream clustering functions) A snapshot of the model parameters.
REFERENCED_MODEL	(Used only by the Hoeffding Tree scoring function) The training model being used for scoring algorithms.
MEMORY_QUOTA	The maximum amount of memory that smart data streaming can use for the model. If the model exceeds the memory quota, the application shuts down, and prompts you to either increase the quota, or clear the model content.

Property	Description
SYNC_POINT	The interval, in seconds (S) or number of rows (N), that determines when smart data streaming backs up model content. If the project shuts down unexpectedly, smart data streaming uses the most recent backup to continue training, scoring, or clustering. Each function syncs with the SAP HANA database for a slightly different purpose: • Training models create backups on sync, which help recover model content after an unexpected shutdown, and retrain the model. • Scoring models create backups on sync, and gather the latest training information. • Clustering models create backups on sync, which help recover model content after an unexpected shutdown.

The STREAMING_ML_MODEL_CONTENT_ table stores the latest model content from PAL functions, including a snapshot of the metadata table, the processed data computed from streaming events, and information on the latest updates to the model:

Field	Description
MODEL_NAME	The name of the model. This value is a primary key.
INDEX	The row index. If the model content is too long to hold in one row, refer to this index. This value is a primary key.
IMPORTANT_METADATA	A snapshot of the latest model metadata. Since metadata changes and updates may affect the validity of model content, refer to this snapshot to identify the source of the changes.
MODEL_CONTENT_VERSION	The version of the model content. The version number increases by 1 every time the algorithm modifies the model content.
MODEL_CONTENT	The latest model content in JSON string format.
LAST_CHANGE_BY	The ID of the user that edited the model most recently.
LAST_CHANGE_AT	A timestamp for the last change to the model.

The STREAMING_ML_MODEL_EVALUATION_ table stores statistics about model performance:

Field	Description
MODEL_NAME	The name of the model. This value is a primary key.
MEASURE_NAME	The name of a model evaluation measure (a means of statistically evaluating the model performance). This value is a primary key.
MEASURE_VALUE	The value of the model evaluation measure.

9.3 Example: Hoeffding Tree Training and Scoring

This example shows Hoeffding Tree training and Hoeffding Tree scoring models used in streaming projects.

The first model in this example uses the Hoeffding Tree training function to analyze streaming census data, including personal details such as age, education, and occupation, to predict a person's income level. Data like this could be very useful to a mortgage specialist, for example, in order to judge an applicant's creditworthiness and predict what kind of mortgage is the most suitable.

First, we create the model `TrainingModel` on the data service `palservice` using the following properties:

- **Description**: training model
- **Machine-Learning Function**: HoeffdingTree Training
- **Input Schema**: integer, string, double, string, integer, string, string, string, string, string
- **Output Schema**: double
- **Memory Quota**: 1024
- **Sync Point**: 5000 N

For the purpose of this example, the values under the parameters tab remain as default.

Then, in the CCL example below, we declare the model `TrainingModel` with its input and output schemas, and the data service name. As you can see, the input schema of the model matches the input schema of `InStream1`. The other properties and parameters of the model are not visible in CCL. The model stream `NewModelStream1` pulls in data from `InStream1`, and applies the analytic function from the model `TrainingModel`. Then, `NewModelStream1` outputs data into an SAP HANA table:

```
DECLARE MODEL TrainingModel
    TYPE MACHINE_LEARNING
    INPUT SCHEMA (
        Column0 INTEGER,
        Column1 STRING,
        Column2 DOUBLE,
        Column3 STRING,
        Column4 INTEGER,
        Column5 STRING,
        Column6 STRING,
        Column7 STRING,
        Column8 STRING,
        Column9 STRING )
    OUTPUT SCHEMA (
        Column0 DOUBLE )
    PROPERTIES
         Dataservice = 'palservice';
CREATE INPUT STREAM InStream1
    SCHEMA
        age INTEGER,
        workclass STRING,
        fnlwgt DOUBLE,
        education STRING,
        numeducation INTEGER,
        maritalstatus STRING,
        occupation STRING,
        relationship STRING,
        race STRING
    );
CREATE OUTPUT STREAM NewModelStream1
    AS EXECUTE MODEL TrainingModel
    FROM InStream1;
```

As the training function is running, a second project leverages the training information through the Hoeffding Tree scoring function. Using the latest model generated from training, the scoring algorithm constantly applies the most recent training data to draw out useful statistical information.

This time, we create a second model, named `ScoringModel`, on the data service `palservice` using the following properties:

- **Description**: scoring model
- **Machine-Learning Function**: HoeffdingTree Scoring
- **Input Schema**: integer, integer, string, double, string, integer, string, string, string, string, string

- **Output Schema**: integer, string, double
- **Memory Quota**: 1024
- **Sync Point**: 5000 N
- **Referenced Model**: TrainingModel

Scoring models have no additional mathematical parameters.

In the example below, we use a very similar setup to the training model. In this project, however, the input schemas in both `Stream1` and the `scoringmodel` gain a leading ID column, and the model output schema gains two more columns, which provide information about model performance. As well, this project contains an optional output stream, `OutStream1`, to further direct the data from the scoring model stream:

```
MODEL ScoringModel
TYPE MACHINE_LEARNING
    INPUT SCHEMA (
        Column0 INTEGER ,
        Column1 INTEGER ,
        Column2 STRING ,
        Column3 DOUBLE ,
        Column4 STRING ,
        Column5 INTEGER ,
        Column6 STRING ,
        Column7 STRING ,
        Column8 STRING ,
        Column9 STRING )
    OUTPUT SCHEMA (
        Column0 INTEGER ,
        Column1 STRING ,
        Column2 DOUBLE )
    PROPERTIES
        dataservice = 'palservice' ;
CREATE INPUT STREAM InStream1
    SCHEMA (
        id INTEGER,
        age INTEGER,
        workclass STRING,
        fnlwgt DOUBLE,
        education STRING,
        numeducation INTEGER,
        maritalstatus STRING,
        occupation STRING,
        relationship STRING,
        race STRING
    );
CREATE OUTPUT STREAM NewModelStream1
    AS EXECUTE MODEL scoringmodel
    FROM InStream1 ;
CREATE OUTPUT STREAM OutStream1
    AS SELECT *
    FROM NewModelStream1 ;
```

9.4 Example: DenStream Clustering

This example shows the DenStream clustering model used in a streaming project.

In this example, the clustering algorithm evaluates streaming age and income statistics to discover patterns that are not immediately apparent to the human eye. By grouping events based on these two attributes, the algorithm learns what the the most common income amounts are for various age groups, and also identifies outlying events, such as a high-income earner in an age group where income is generally low.

First, we create the model `ClusteringModel` on the data service `palservice` using the following properties:

- **Description**: clustering model
- **Machine-Learning Function**: DenStream Clustering
- **Input Schema**: integer, double, integer
- **Output Schema**: integer, integer, string, double, integer, string, string
- **Memory Quota**: 1024
- **Sync Point**: 5 N

Then, we set the parameters as follows:

- **INIT N**: 50
- **Epsilon**: 16
- **Beta**: 0.2
- **Mu**: 10
- **Lambda**: 0.25
- **Clustering Condition**: 50
- **Max. Number of Categories**: 0

Then, in the CCL example below, we declare the model `ClusteringModel` with its input and output schemas, and the data service name. As you can see, the input schema of the model matches the input schema of `InStream1`. The other properties and parameters of the model are not visible in CCL. The model stream `NewModelStream1` pulls in data from `InStream1`, and applies the analytic function from the model `ClusteringModel`. Then, `OutputStream1` selects and outputs data into an SAP HANA table:

```
DECLARE MODEL ClusteringModel
    TYPE MACHINE_LEARNING
    INPUT SCHEMA (
        Column0 INTEGER,
        Column1 DOUBLE,
        Column2 INTEGER )
    OUTPUT SCHEMA (
        Column0 INTEGER ,
        Column1 INTEGER ,
        Column2 STRING ,
        Column3 DOUBLE ,
        Column4 INTEGER ,
        Column5 STRING ,
        Column6 STRING )
    PROPERTIES
         Dataservice = 'palservice';
CREATE INPUT STREAM InStream1
    SCHEMA (
        id INTEGER,
        income DOUBLE,
        age INTEGER );
CREATE OUTPUT STREAM NewModelStream1
    AS EXECUTE MODEL ClusteringModel
    FROM InStream1;
CREATE OUTPUT STREAM OutputStream1
    AS SELECT *
    FROM modelStream1;
```

10 Streaming Lite

Streaming lite is a to-the-edge component designed to remotely deploy streaming projects. Streaming lite is relevant if you wish to deploy projects on remote gateway devices - it is not required as part of a standard smart data streaming installation.

Elements of Streaming Lite

In order to run and test streaming projects, streaming lite includes the following:

- streamingclusteradmin
- hdbstreamingserver
- streamingclusterutil
- streamingcnc
- streamingcompiler
- streamingcompiler
- streamingconvert
- streamingencrypt
- streamingmonitor
- streamingproject executable (and required libraries)
- streamingprojectclient executable (and required libraries)
- streamingsubscribe
- streamingupload
- Streaming Web Output adapter
- Database Input and Output adapters
- C and Java SDKs

Note

For the executables, all options are the same as described for smart data streaming, except: `-P <host:command port>`. Because streaming lite does not use a clustering architecture, use `command port` directly to connect to streaming lite. There is no concept of workspace or project in streaming lite.

Note

The Database Input and Output adapters support connections only to SQL Anywhere when used with streaming lite. On streaming lite for ARM, the Database adapters do not support conversion between the smart data streaming `time` datatype and the SQL Anywhere `TIME` datatype.

Unsupported Streaming Elements

Because streaming lite is a stand-alone component and offers a remote environment for deployment rather than project development, it does not support the following:

- .NET SDK
- SAP HANA studio, the SAP HANA Streaming Development or SAP HANA Streaming Run-Test perspectives
- SAP HANA cockpit
- Internal, external or toolkit framework adapters (streaming lite only supports the Streaming Web Output adapter and the Database Input and Output adapters)
- Continuous Computation Language (CCL) Reference
- CCL Database Connectivity

> **Note**
>
> All other CCL features are supported by streaming lite; however with CCL Date and Time conversion for 32bit ARM devices, there is a known limitation with date conversion functions for dates outside of the range 1902 to 2037.

Streaming Lite Platform and Installation

Streaming lite is optional and not required as part of a standard smart data streaming installation. The streaming lite package is downloadable as a separate component only. As a separate component, streaming lite can run on the following platforms, independent of the platform on which you are running SAP HANA smart data streaming:

Platform for Streaming Lite	Product Build Environment	Product Build Compiler	Supported Operating System	Minimum Requirements
Linux -x86 64 bit	RHEL 5.5	GCC 4.6.2	RHEL 5.5 to 5.10	512MB RAM, 1GB HDD, 1 core CPU
Linux-ARM 32 bit (Version ARMv6l or higher)	Ubuntu 14.04 LTS x86_64	GCC (Crosstool-NG Linaro-1.13.1-4.8-2014.01 - Linaro GCC 2013.11) 4.8.3 20140106 (prerelease)	Raspbian 3.12.35 (wheezy)	Raspberry-pi device A + and above with default configuration

You can download the Linux and/or the ARM versions of the streaming lite tarball, provided that you have an SAP HANA smart data streaming license for any platform. For detailed instructions, see *Installing the Streaming Lite Specialized Component* in the *SAP HANA Smart Data Streaming: Installation and Update Guide*.

In this section:

Streaming Lite Configuration [page 244]
Streaming lite is designed to deploy streaming projects on remote gateway devices: you can use smart data streaming for CCL development and testing, and then deploy the project in streaming lite.

Related Information

SAP HANA Smart Data Streaming: Installation and Update Guide

10.1 Streaming Lite Configuration

Streaming lite is designed to deploy streaming projects on remote gateway devices: you can use smart data streaming for CCL development and testing, and then deploy the project in streaming lite.

The mechanism that client applications can use to authenticate to streaming lite is the Pluggable Authentication Module (PAM) package in Linux. Streaming lite only supports PAM based authentications. Configure PAM before you can use it with streaming lite.

Streaming lite supports Secure Socket Layer (SSL) connections over the network to ensure the privacy of communication between the client applications and streaming lite. SSL connections use cryptographic keys and certificates to implement secure data transfer. Streaming lite supports SSL (TLSv1.2).

In this section:

10.1.1 Configuring SSL Connections for Streaming Lite

Streaming lite supports Secure Socket Layer (SSL) connections over the network to ensure the privacy of communication between the client applications and streaming lite. SSL connections use cryptographic keys and certificates to implement secure data transfer.

Context

Streaming lite supports SSL (TLSv1.2). The streaming lite distribution package does not come with pre-generated SSL keys and certificates. Instead, generate key and certificate using the openssl tool.

Procedure

1. Create a self-signed certificate by using the command :

   ```
   openssl req -new -text -out server.req -days n
   ```

   where `server.req` is a filename for the certificate and `n` is the number of days that the certificate will be valid.
2. As prompted, enter a passphrase (at least four characters long), and other information (such as company name and location). The challenge password can be left blank.

   The program generates a key that is passphrase protected.
3. Remove the passphrase by using the command:

   ```
   openssl rsa -in privkey.pem -out server.key
   rm privkey.pem
   ```

   where server.key is a filename for the key files. Use the passphrase you previously entered to unlock the existing key.
4. Turn the certificate into a self-signed certificate by using the command:

   ```
   openssl req -x509 -in server.req -text -key server.key  -out server.crt -days
   n
   chmod og-rwx server.key
   ```

   where `server.key` is the name of the key file and `n` is the number of days that the certificate will be valid.
5. Copy the `server.key` and `server.crt` to the key directory that will be referenced when you start streaming lite with the `-e` option.

10.1.2 Configuring PAM Authentication for Streaming Lite

The mechanism that client applications can use to authenticate to streaming lite is the Pluggable Authentication Module (PAM) package in Linux. Streaming lite only supports PAM based authentications. You must configure PAM before you can use it with streaming lite.

Context

In PAM authentication, client applications connecting to streaming lite through the Command and Control interface, or the Gateway interface, must supply a user name and password before they can issue commands or requests. PAM allows you to integrate with OS, LDAP and Kerberos security providers. Streaming lite uses PAM to verify that the username and password are correct. It is your responsibility as administrator to specify whether this password check should be made against the Linux password file, an LDAP server, or a NIS system.

Procedure

1. To configure authentication on Linux, log in as root.
2. Create the file `/etc/pam.d/sp`. The contents of this file determine the type of authentication required.

   For LINUX password authentication, the contents of the file `/etc/pam.d/sp` should be:

   ```
   auth required pam_permit.so
   auth required pam_warn.so
   auth required pam_unix.so
   ```

   The `/etc/shadow` file must be readable by the user who starts streaming lite.

   For LDAP authentication, the contents of the `/etc/pam.d/sp` file should be:

   ```
   auth required pam_permit.so
   auth required pam_warn.so
   auth required pam_ldap.so
   ```

10.2 Deploying a Project in Streaming Lite

Streaming lite is a remote deployment environment rather than a development environment. Once you have developed a project using smart data streaming and you are ready to do so, you can deploy the CCX and CCR project files in the streaming lite environment.

Prerequisites

- Create the project using smart data streaming. Ensure you have not used adapters or features unsupported by streaming lite, as outlined in Streaming Lite [page 242].
- Compile and test the project using streaming lite to ensure that it runs.

Procedure

1. Decompress the streaming lite tarball:

```
gunzip streaming_lite_1.0.100.00_linuxarm32.tgz
```

2. Upload the `.tar` file to the streaming lite server:

```
scp streaming_lite_1.0.100.00_linuxarm32.tar <username>@<hostname>:<directory>
```

3. Untar the uploaded file using the command:

```
tar xvf streaming_lite_1.0.100.00_linuxarm32.tar -C <target-directory>
```

4. Set the STREAMING_HOME environment variable to the location of the extracted `install` folder.

```
export STREAMING_HOME=<path-to-install-folder>
```

5. Upload the project CCX and CCR files to the streaming lite extracted folder.
6. Start the project using `streamingproject`:

> Note
>
> If the project uses a data service with encrypted credentials, run `streamingproject` from the location of the cipher key file.

```
./streamingproject --ccx <ccx-file> --command-port <port> --host <hostname>
```

In this section:

streamingproject [page 248]
The streamingproject executable is used to start streaming lite. This server executable cannot be used with components of smart data streaming.

Starting or Stopping Streaming Lite [page 249]
Use streamingproject to start or stop streaming lite.

10.2.1 streamingproject

The `streamingproject` executable is used to start streaming lite. This server executable cannot be used with components of smart data streaming.

Prerequisite

Set the STREAMING_HOME environment variable to the folder where you extracted the streaming lite tar file.

1. From a command shell, go to the directory containing the streaming lite bin folder.
2. Use the command:

```
export STREAMING_HOME=$PWD
```

Syntax

```
streamingproject  [<options...>]
```

Options

--ccr

Identifies the ccr file name including path.

--ccx

Identifies the ccx file name including path.

--command-port

Identifies the command port which the streaming project monitors.

-e

(Optional) Specifies the folder where Secure Socket Layer (SSL) keys are stored.

> Note
>
> Generate SSL keys for streaming lite before using this option.

-h

(Optional) Prints a list of possible options on the screen along with a brief explanation for each option.

Alias: `--help`.

--host

Identifies the hostname of the device where the streaming project runs. If not specified, it runs in localhost, which may be inaccessible externally.

-V

(Optional) Instructs the streaming project to use the Pluggable Authentication Module (PAM) security check at login.

> **Note**
>
> Configure PAM for streaming lite before using this option.

Usage

```
 ./streamingproject --ccx sensorproject.ccx --command-port 32222 --host
raspberrypi
```

10.2.2 Starting or Stopping Streaming Lite

Use `streamingproject` to start or stop streaming lite.

Starting Streaming Lite with SSL

To start streaming lite in a mode that supports Secure Socket Layer (SSL) connections, use the `-e` option with the `streamingproject` command, and specify the location of the directory containing the server private key file (`server.key`), and certificate (`server.crt`):

```
streamingproject --ccx <ccx file name> --command-port <command port> --host
<hostname> -e /home/sdslite/etc/keys
```

> **Note**
>
> Generate SSL keys for streaming lite to be able to use the `-e` option.

Starting Streaming Lite with PAM

To start streaming lite with Pluggable Authentication Module (PAM) authentication, use the `-V` option with the `streamingproject` command:

```
streamingproject --ccx <ccx file name> --command-port <command port> --host
<hostname>  -V pam
```

> **Note**
>
> Configure PAM for streaming lite before using the `-V` option.

Stopping Streaming Lite

To stop streaming lite, use the command:

```
./streamingprojectclient [-c username: password] -p <hostname of streaming
lite>:<command port of streaming lite> stop
```

10.3 Streaming Web Output Adapter

The Streaming Web Output adapter sends data from a source project deployed in streaming lite to a destination project in smart data streaming via the Streaming Web Service.

The Streaming Web Output adapter supports the same platforms and operating systems as streaming lite. Before deploying the source project in streaming lite, develop the project and configure the adapter using the smart data streaming plugin for SAP HANA studio. The adapter configuration (`.cnxml`) file is included on all platforms.

Each instance of the adapter sends data from an output stream in the streaming lite source project to a target stream or window in the smart data streaming destination project. You can specify how columns map from the source stream to the destination stream using the `Destination Column Names` property. When specifying column names, keep in mind that:

- Columns map from the source stream to the target stream in the order they are listed.
- Columns not in the column name property contain null values.
- If you specify more column names than there are columns in the schema of the source stream, the adapter ignores the property.

In the following example, when sending data from `source_stream` to `target_stream`, col1 and col2 send data to colB and colC, respectively, and colA is empty:

```
CREATE INPUT STREAM source_stream SCHEMA ( col1 integer, col2 integer, col3
integer );
ATTACH OUTPUT ADAPTER myAdapter TYPE sws_out TO source_stream
PROPERTIES
    protocol = 'rest',
```

```
    columnNames = 'colB, colC',
    clusterAuthenticationData = 'sybase:sybase',
    swsHost = 'cepamd64linux',
    swsPort = '9091',
    destinationWorkspaceName = 'default',
    destinationProjectName = 'destination',
    destinationStreamName = 'target_stream';
```

```
CREATE INPUT STREAM target_stream SCHEMA ( colA integer, colB integer, colC
integer );
```

If you do not specify the column names, the schema of the streaming lite output stream must match the schema of the destination stream in smart data streaming. If the column names are different, the data is not sent, and the Streaming Web Service inserts rows with null values.

Error messages for the adapter are in the project trace file. If an error occurs, the project does not go down. If the connection goes down, the project attempts to reconnect and continue sending data.

The Streaming Web Output adapter must be used with the Streaming Web Service. The adapter supports both REST and WebSocket protocols, however, it only supports guaranteed delivery when the Streaming Web Service is running in REST mode. For more information, see *Streaming Web Service* in the *SAP HANA Smart Data Streaming: Adapters Guide*.

In this section:

10.3.1 Streaming Web Output Adapter Properties

Set properties for the Streaming Web Output adapter in the adapter properties dialog of smart data streaming in SAP HANA studio.

The Streaming Web Output adapter can be used when the source project is deployed on streaming lite.

Property Label	Description
PropertySet	Property ID: `propertyset` Type: `string` (Advanced) Specify the name of the property set from the project configuration file that you want to use. If you set the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values in the ATTACH ADAPTER statement. No default value.
Protocol	Property ID: `protocol` Type: `choice` (Required) Specify which protocol you want the adapter to use to send data. Accepted values are rest and websocket. Default value is `rest`.
Destination Column Names	Property ID: `columnNames` Type: `string` (Advanced) Specify the column names of the destination stream as a comma-delimited list. For example, "col1,col2,col3". When specifying column names: • Columns map from the source stream to the target stream in the order they are listed • Columns not in the column name property contain null values • If you specify more column names than the number of columns in the source stream schema, the adapter ignores the property If you do not specify the column names, the source stream schema in the streaming lite project must match the destination stream schema in the smart data streaming project. No default value.
Destination Cluster Authentication Data	Property ID: `clusterAuthenticationData` Type: `string` (Required) Specify the credentials for the smart data streaming cluster containing the destination smart data streaming project, using the format username:password. For example, sap:sap. If the credentials are encrypted, set the `encrypted` value to true. No default value. See the *Encrypting the Streaming Web Service Credentials* topic for instructions on running a streaming lite project with the Streaming Web Output adapter using encrypted credentials.

Property Label	Description
SWS Host	Property ID: `swsHost` Type: `string` (Required) Specify the hostname for the Streaming Web Service. No default value.
SWS Port	Property ID: `swsPort` Type: `string` (Required) Specify the port number for the Streaming Web Service. Default value is 8000.
Destination Workspace Name	Property ID: `destinationWorkspaceName` Type: `string` (Required) Specify the workspace containing the destination smart data streaming project. Default value is `default`.
Destination Project Name	Property ID: `destinationProjectName` Type: `string` (Required) Specify the smart data streaming project containing the destination stream or window. Default value is `destination`.
Destination Stream Name	Property ID: `destinationStreamName` Type: `string` (Required) Specify the name of the destination stream or window. Default value is `NEWSTREAM`.
Use SSL	Property ID: `useSSL` Type: `boolean` (Advanced) Specify if the Streaming Web Service uses SSL. Default value is `false`.
Absolute Trust File Path	Property ID: `trustCert` Type: `string` (Advanced) Specify the absolute path to the trust file. No default value.

Property Label	Description
Runs Adapter GD Mode	Property ID: `enableGDMode` Type: `boolean` (Advanced; use only with REST requests) Specifies whether the adapter runs in guaranteed delivery (GD) mode. GD ensures that data continues to be processed in the case that the smart data streaming server fails, or the destination (third-party server) fails or does not respond for a long time. See *Guaranteed Delivery* in the *SAP HANA Smart Data Streaming: Developer Guide* for details on enabling GD for your project. The default value is `false`.
Enable the Adapter Internal Cache Mode	Property ID: `enableGDCache` Type: `boolean` (Advanced; use only with REST requests) If set to true, only rows that can be recovered (that is, checkpointed) by the smart data streaming server on restart are sent to the Streaming Web Service. Other rows are cached internally by the adapter. When this option is enabled, you may see a significant increase in latency depending on how frequently the attached stream delivers checkpoint messages. Streams support three modes related to GD: GD not supported, GD supported, and GD supported with checkpoint messages. This setting is ignored if the attached stream does not support GD and does not deliver checkpoint message. The default value is `true`.
GD Batch Size	Property ID: `gdBatchSize` Type: `int` (Advanced; use only with REST requests) Specify after how many message blocks (transactions or envelopes) the adapter issues a GD commit to the stream to which the adapter is attached. The default value is 10. Increasing this value improves performance at the risk of increasing latency, caused by delivering a larger number of duplicate rows to SWS in the event the server is restarted. Increasing this value also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure.
Max Reconnection Attempts	Property ID: `reconnectAttempts` Type: `int` (Advanced; use only with REST requests) The maximum number of times the adapter attempts to reconnect to the Streaming Web Service after failing to send data. Use -1 to retry an unlimited number of times. Default value is 5.

Property Label	Description
Reconnection Wait Interval (in ms)	Property ID: `reconnectIntervalMsec` Type: `int` (Advanced; use only with REST requests) The amount of time (in milliseconds) the adapter waits between attempts to reconnect to the Streaming Web Service. Default value is 10000.

10.3.2 Sending Data to a Destination

When a project deployed on streaming lite publishes data, you can send this data to a SAP HANA smart data streaming streaming project, using the Streaming Web Output adapter.

Context

The Streaming Web Output adapter must be used with the Streaming Web Service. For more information, see the *Streaming Web Service* section in the *SAP HANA Smart Data Streaming: Adapters Guide*.

Procedure

1. Ensure that the SAP HANA smart data streaming cluster is running.
2. Ensure that the project where you want to send the streaming lite data is running in the SAP HANA smart data streaming cluster.
3. Ensure that the Streaming Web Service is running.
4. Attach the Streaming Web Output adapter to the output stream of the project deployed on streaming lite, and set the properties appropriately.
5. Publish the data in the project deployed on streaming lite.

   The data is visible in the project running in the SAP HANA smart data streaming cluster.

10.3.3 Encrypting the Streaming Web Service Credentials

To run a streaming lite project with the Streaming Web Output adapter using encrypted credentials, generate a cipher key file to encrypt the Streaming Web Service credentials.

Procedure

1. Set $STREAMING_HOME to the location of your streaming lite installation:

```
export STREAMING_HOME=<streaming-lite-install-dir>
```

2. Generate a `cipher.key` file:

```
$STREAMING_HOME/bin/streamingencrypt -c > cipher.key
```

3. Use the key file to encrypt your credentials:

```
$STREAMING_HOME/bin/streamingencrypt -e cipher.key -t <username>:<password>
```

4. In the Streaming Web Output adapter properties section of the project configuration (`.ccr`) file, set `clusterAuthenticationData` to the encrypted credentials and set the `encrypted` value to true:

```
<Property encrypted="true" name="clusterAuthenticationData"><encrypted-
credentials></Property>
```

5. Copy the `cipher.key` file to the `$STREAMING_HOME` folder in your streaming lite installation.
6. Copy the project configuration file to the location of the `.ccx` file in your streaming lite installation.

Next Steps

To start the project, run the `streamingproject` utility from `$STREAMING_HOME`. The `cipher.key` file must be in the location from which you run `streamingproject`.

10.4 Configuring an ODBC Connection for Streaming Lite

Configure an ODBC connection from streaming lite to an SAP SQL Anywhere database.

Prerequisites

SAP SQL Anywhere is installed.

Procedure

1. Append the location of the SQL Anywhere ODBC driver to the LD_LIBRARY_PATH environment variable. For example, if SQL Anywhere is installed at /home/pi, execute the following command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/pi/SQLA/lib32
```

2. Create a link between `$STREAMING_HOME/lib/libodbc.so.1` and the SQL Anywhere ODBC driver manager:

```
ln -s /home/pi/SQLA/lib32/libdbodm16.so.1 $STREAMING_HOME/lib/libodbc.so.2
```

3. Define the DSN in the `~/.odbc.ini` file. Create this file if it does not already exist:

```
[SQLA]
Description = SQL Anywhere 16 DB
Driver=/home/pi/SQLA/lib32/libdbodbc16.so
Userid=DBA
Password=sql
DatabaseFile=/home/pi/SQLA/sample.db
Port=2638
ServerName=sample
```

4. Define the data service in the `$STREAMING_HOME/bin/service.xml` file:

```
<Service Name="SQLAService" Type="DB">
    <Parameter Name="DriverLibrary">streamingdbodbc_lib</Parameter>
    <Parameter Name="DSN">SQLA</Parameter>
    <Parameter Name="User">DBA</Parameter>
    <Parameter Name="Password">sql</Parameter>
</Service>
```

In this section:

Encrypting Data Service Credentials [page 257]
Use the streamingencrypt utility to encrypt passwords in the `service.xml` file.

10.4.1 Encrypting Data Service Credentials

Use the streamingencrypt utility to encrypt passwords in the `service.xml` file.

Procedure

1. Set $STREAMING_HOME to the location of your streaming lite installation:

```
export STREAMING_HOME=<streaming-lite-install-dir>
```

2. Generate a cipher key:

```
$STREAMING_HOME/bin/streamingencrypt -c > cipher.key
```

3. Use the cipher key to encrypt the password:

```
$STREAMING_HOME/bin/streamingencrypt -e cipher.key -t <password>
```

4. Paste the encrypted password into the `$STREAMING_HOME/bin/service.xml` file, and set the `encrypted` attribute to true:

```
<Service Name="SQLAService" Type="DB">
    <Parameter Name="DriverLibrary">streamingdbodbc_lib</Parameter>
    <Parameter Name="DSN">SQLA</Parameter>
    <Parameter Name="User">DBA</Parameter>
    <Parameter Name="Password" encrypted="true"><encrypted-password></
Parameter>
</Service>
```

Next Steps

When starting a project that uses encrypted data service credentials, make sure the `cipher.key` file is in the location from which you run streamingproject.

10.5 Using the Streaming Lite Sample Project

The streaming lite sample project allows you to start streaming lite with the `sensorproject.ccx` file included in the smart data streaming examples folder `$STREAMING_HOME/examples/lite`.

Context

This example only works with streaming lite.

At minimum, two columns are required in the destination stream in order to successfully send data. The first column contains an auto-generated timestamp. The second column contains the actual value to be sent, such as a temperature reading. Other columns are optional. If you want to send them any data then it should be provided as command arguments.

```
(ts long, value decimal(18, 8), uuid string, type string, unit string, longitude
string, latitude string, location string, other string)
```

This example does not support SSL.

Procedure

1. Compile the example:

```
make -f Makefile.lite
```

2. Set STREAMING_HOME to the folder where you have installed streaming lite.
3. Enter the command:

```
$STREAMING_HOME/bin/streamingproject --ccx sensorproject.ccx --command-port
9094 &
```

   If streaming lite is running on a host and port other than localhost:9094, you can change this with the command line option `uri`.
4. (Optional) This command sequence expects that there is no authentication in streaming lite. If you started streaming lite with the `-V` option then use the `creds` options to provide the username:password.
5. Once the sample project starts, send temperature sensor data:

```
 ./sendsignals --command="echo 70" --type="temperature" --unit=F
```

   `sendsignals` executes the command provided as part of the `--command` argument to generate a number. You can use `echo 70` to generate the value 70, or use any executable, such as the sensor driver executable to get the temperature reading.

   This command expects an input stream name called `signals`. You can also supply this through the command line using `stream`.

   At minimum, it expects that the input stream will have two columns: one long and one decimal. It automatically generates a timestamp in seconds for the first column. The second column is populated with the data provided in the `command` option. Data for other columns need to be provided as part of the command line argument but are optional.
6. To display usage:

   `--help sendsignal` executes the command provided as part of the `--command` argument to generate a number.
7. (Optional) Use the subscriber tool to subscribe the stream at Streaming Lite:

```
$STREAMING_HOME/bin/streamingsubscribe --command-port 9094 -s signalout
```

8. (Optional) `sensorsimulator` is a random number generator within a range, the default being 1 to 120, which you can provide as an argument. Use this executable to simulate a sensor. To generate a random number between 20 and 90:

```
./sendsignals --command="./sensorsimulator 20 90" --type="temperature" --
unit=F --repeat=-1
```

11 SAP PowerDesigner for SAP HANA Smart Data Streaming

SAP HANA smart data streaming users create and manipulate the Streaming Schema using SAP PowerDesigner®.

PowerDesigner is a powerful modeling tool. SAP HANA smart data streaming users can use it to develop physical data models as well as the logical data models that define the Streaming Schema.

In this section:

11.1 Getting Started

SAP PowerDesigner® is a tool for creating and manipulating Streaming Schema. Optionally, it can be used with physical data models.

This section is intended for database and application development staff, and for SAP Professional Services representatives, customer IT support, and other technical personnel who set up and administer PowerDesigner. It includes information you need to understand, model, and modify logical schema definitions and physical database structure when developing schema.

In this section:

11.1.1 Data Modeling Scenarios

Integrated modeling supports efficient schema definition and database design, and consistent production deployments.

Use the Streaming Schema and extensions to:

- Model schema in the Streaming Schema model, a PowerDesigner logical data model.
- Convert Streaming Schema logical data models to SAP HANA, SAP ASE, or SAP IQ physical data models.
- Convert existing SAP HANA, SAP ASE, and SAP IQ physical data models to Streaming Schema logical data models.
- Import schema definitions defined in a CCL file into Streaming Schema models.
- Export schema definitions from Streaming Schema models into a CCL file.
- Validate a model using custom checks for Streaming Schema, in addition to the standard PowerDesigner checks.
- Analyze the impact of changes to schema, a model, or a database table on all components in the integrated model.

The corresponding adapter (SAP IQ, HANA, ASE) schema must match the SAP IQ, HANA, and ASE database schema for all tables in which data is inserted. After you make changes, you can use PowerDesigner to produce a set of data definition language (DDL) statements directly from the physical data models (SAP IQ, HANA, and ASE). PowerDesigner saves the DDL in a SQL script that you can run to generate the tables and other objects for the target databases.

DDL generation does not require use of the extended modeling feature.

11.1.2 Sample PowerDesigner Project

A sample project supports integrated modeling.

You can install a PowerDesigner sample project that includes:

- A sample Streaming Schema logical model.
- SAP IQ, SAP HANA, and SAP ASE physical data models.

In this section:

11.1.2.1 Opening the Sample Project

Open the sample model from the sample project.

Procedure

1. Choose **Start** > **Programs** > **Sybase** > **PowerDesigner 16**.
2. In the Welcome dialog, under Getting started, choose **Open Model or Project**.

   If you are not a first-time user, you may see different options in the Welcome dialog, based on your previous work in PowerDesigner.
3. Browse to the sample project in `%PowerDesigner 16%\Examples\Streaming\Streaming.prj` and choose **Open**.

   PowerDesigner opens a workspace for the Streaming Schema sample project.
4. Double-click the **Streaming** project ( ).
   The sample project opens with the sample Streaming Schema model, SAP IQ model, SAP ASE model, SAP HANA model, and the Model Relationship Diagram in the Browser view.

11.1.3 Learning More About PowerDesigner

For more information on using PowerDesigner, press **F1** to open the online help, or see the PowerDesigner online product documentation.

11.2 Data Model

PowerDesigner includes a logical data model for Streaming Schema and three physical data models for the SAP IQ, SAP HANA, and SAP ASE databases.

The indexes for both physical data models are database-specific and are defined individually. You can open, view, modify, and extend the data models using PowerDesigner.

In this section:

The Streaming Schema model represents market data in a logical data model independent of any data store.

Finding an Object in a Diagram [page 263]

Locate any object with a symbol in a diagram or among several diagrams. Objects without graphical symbols, such as domains, are not shown in diagrams.

Data Model Tables [page 264]

The following table lists all data model tables in the Market Data diagrams with their code names and descriptions.

Extensions [page 265]

Extensions (`.xem` files) provide a means to customize and extend PowerDesigner metaclasses, parameters, and generation. Use extended models to store additional information, or to change model behavior.

11.2.1 Streaming Schema Logical Data Model

The Streaming Schema model represents market data in a logical data model independent of any data store.

The Streaming Schema logical model represents the building of schema and the databases parsing schema and storing them.

The Streaming Schema model contains a definition for each schema. The schema definitions are contained in the Market Data diagram in the sample Streaming Schema model. Adding schema to the diagram is optional.

To create a new Streaming Schema model, you can:

- Create it from scratch using the Streaming Schema Model category.
- Create it from scratch using the `StreamingSchema.xem` file to extend the model during or after creation.
- Generate it from an SAP IQ, SAP HANA, or SAP ASE physical data model.

11.2.2 Finding an Object in a Diagram

Locate any object with a symbol in a diagram or among several diagrams. Objects without graphical symbols, such as domains, are not shown in diagrams.

Procedure

Right-click an object in the Browser and select **Find in Diagram**.

11.2.3 Data Model Tables

The following table lists all data model tables in the Market Data diagrams with their code names and descriptions.

Table name	Code	Description
Bond History	BOND_HISTORY	Stores bond historical data, one record per each trading date. The data includes daily price and yield values (open/close, high/low), trade volume (number of bonds traded), and so on, for each bond.
Bond Quote	BOND_QUOTE	Stores real-time (intraday) quote data. Each quote record includes a yield, bid/ask price, and size (in other words, a number of bonds offered at a bid/ask price).
Bond Trade	BOND_TRADE	Stores real-time (intraday) trade data. Each trade record includes a bond's price and yield and a transaction's size (number of bonds traded).
Dividend Event	DIVIDEND_EVENT	Stores information on a dividend payment event when a shareholder receives a certain payment for each share of stock owned. The dividend amount is commonly defined as a certain percentage of a share price but can also be specified as a monetary amount. The Monetary or Percentage Indicator (MOP_INDICATOR) column indicates how the dividend amount is defined.
Index History	INDEX_HISTORY	Stores the index's historical data, one record per each trading date. The data includes the index's daily values (open/close, high/low) and trade volume.
Index Intraday	INDEX_INTRADAY	Stores the index's real-time (intraday) data that shows its value movements during a trading day. Each data point includes an index value and trade volume.
Mutual Fund History	MUTL_FUND_HIST	Stores the historical data for a mutual fund, one record per each trading date. The data includes a trade date and price.
Option History	OPTION_HISTORY	Stores the options historical data, one record per each trading date. The data includes options daily price (open/close, high/low), trade volume (number of contracts traded), and so on.
Option Quote	OPTION_QUOTE	Stores the options real-time (intraday) quote data. Each quote record includes a bid/ask price, size (number of contracts offered at a bid/ask price), and so on.

Table name	Code	Description
Option Trade	OPTION_TRADE	Stores the options real-time (intraday) trade data. Each trade record includes a trade's price, size (number of contracts traded), and so on.
Split Event	SPLIT_EVENT	Stores information on a stock split event when the number of outstanding shares of a company's stock is increased and the price per share is simultaneously decreased so that proportionate equity of each shareholder remains the same. The split is characterized by a split factor; a factor of 0.5 indicates that the number of shares is increased two times and that the share price is decreased two times. In a less common reverse split, the number of shares is decreased and the price per share is increased in a similar manner; a split factor of 2 indicates that the number of shares is decreased two times and that the share price is increased two times.
Stock History	STOCK_HISTORY	Stores the stock historical data, one record per each trading date. The data includes stocks daily prices (open/close, high/low) and trade volume (number of shares traded).
Stock Quote	STOCK_QUOTE	Stores the stocks' real-time (intraday) quote data. Each quote record includes a bid/ask price and corresponding size values (in other words, a number of shares offered at bid/ask price).
Stock Trade	STOCK_TRADE	Stores the stocks' real-time (intraday) trade data. Each trade record includes a transaction's price and size (in other words, a number of shares traded).

11.2.4 Extensions

Extensions (`.xem` files) provide a means to customize and extend PowerDesigner metaclasses, parameters, and generation. Use extended models to store additional information, or to change model behavior.

PowerDesigner provides four `.xem` files:

- `StreamingSchema.xem` – extensions for an logical data model. Contains rules and code that let you model Streaming Schema in a PowerDesigner logical data model.
- `IQ.xem` – extensions for a physical data model. Contains only transformation rules needed to convert a Streaming Schema definition to an SAP IQ table definition, in an SAP IQ model.
- `ASE.xem` – extensions for a physical data model. Contains only transformation rules needed to convert a Streaming Schema definition to an ASE table definition, in an ASE model.

- `HANA.xem` – extensions for a physical data model. Contains only transformation rules needed to convert a Streaming Schema definition to an SAP HANA table definition, in an SAP HANA model.

When you use the models provided with PowerDesigner, the extensions are present. When you create a new model using the Streaming Schema model category set, extensions are applied automatically.

When you create a new model without using the Streaming Schema model categories, or when you have an existing model, you can extend it using the PowerDesigner tools and Streaming extension files.

In this section:

11.2.4.1 Category Set

You can set the Streaming category set to create any Streaming model type.

The Streaming model category set includes Streaming Schema, SAP IQ, SAP HANA, and SAP ASE categories. To create new models from this category set, enable the categories in PowerDesigner. You can either merge the Streaming categories with others that you use, or change PowerDesigner to use only the Streaming categories.

Once you set up the Streaming category set, you can create any Streaming model type and extend it with the appropriate extension.

The `Streaming.mcc` file, installed with the extensions, defines the Streaming categories.

11.2.4.2 Schema Definitions

A schema definition in the Streaming Schema model represents a data stream in SAP HANA smart data streaming.

The sample Streaming Schema model contains a schema definition for each market data table. You can customize any schema definition, or create a new one.

To create a new schema in the Streaming Schema model, you can either:

- Create a schema in PowerDesigner, and then generate a CCL file from it, or;
- Import schema definitions that are defined in a CCL file.

Each schema definition contains:

- Identifiers, which associate schemas with columns that are keys in the associated table.

- Attributes, which associate schemas with a destination column name in the SAP IQ, SAP HANA, and SAP ASE databases with length and precision where appropriate, lookup table and column information for columns that are foreign keys, and descriptive notes.

In this section:

11.2.4.2.1 Sample Schema Definition List

Sample schema definitions correspond to the Market Data diagram provided with PowerDesigner. While each schema appears in the SAP IQ, SAP HANA, and SAP ASE Market Data diagram, not every table in that diagram is a schema.

- Bond History
- Bond Quote
- Bond Trade
- Dividend Event
- Index History
- Index Intraday
- Mutual Fund History
- Option History
- Option Quote
- Option Trade
- Split Event
- Stock History
- Stock Quote
- Stock Trade

11.2.4.3 Impact and Lineage Analysis

PowerDesigner provides powerful tools for analyzing the dependencies between model objects.

When you perform an action on a model object, in a single operation you can produce both:

Impact Analysis	to analyze the effect of the action on the objects that depend on the initial object.
Lineage Analysis	to identify the objects that influence the initial object.

These tools can help you answer questions such as:

- If I change the precision on a column in my SAP ASE model which I generated from the streaming schema model, what table columns in my SAP IQ or SAP HANA model must also change, and what schema are affected?

- Which schema fields influence each column in my SAP ASE, SAP HANA, and SAP IQ models?
- If I delete a column from my SAP IQ model, what is the impact on tables and columns in my SAP ASE and SAP IQ models, and what schema definitions must change in my streaming schema model?

11.3 Extended Model Setup

Your installer automatically sets up the use of extensions.

To apply extensions automatically to new models, set up and use the Streaming Schema model category set.

To integrate existing PDMs with the Streaming model, extend the models by attaching the appropriate extension file.

In this section:

11.3.1 Extending an Existing Model

Attach extensions to any SAP IQ, SAP HANA, or SAP ASE physical data model, or to a logical data model that was generated from the Streaming physical data model but not extended.

Procedure

1. Open the model you want to extend.
2. From the PowerDesigner main menu, choose **Model** > **Extended Model Definitions**.

   > Tip
   >
   > If **Extended Model Definitions** is not in the menu, make sure that the extensions file is unzipped in the folder where PowerDesigner is installed.

3. Click **Import an Extended Model Definition**.

   A list shows available extensions that have not been applied to this model.

4. Select the correct model extension and choose **OK**.
   For example, to extend an ASE physical data model, choose **ASE**.
5. In the **List of Extended Model Definitions** dialog, choose **OK** to extend the model.

Results

PowerDesigner applies the Streaming extensions to the model. No other changes are made. For example, a generic logical data model is not transformed to a Streaming Schema model simply by adding the extensions.

11.3.2 Setting Up the Model Category Set File

Set up PowerDesigner to use the Streaming category set for new models.

Context

PowerDesigner can display only one set of categories in the New Model dialog. While not required, using the Streaming category makes it easier to develop models for use with SAP HANA smart data streaming.

Procedure

- Decide which option you want to use to create new models:

Option	Action required
Only the installed Streaming category set	Change categories
Streaming category set merged with existing categories	Merge Streaming categories
Neither	Manually extend any models

In this section:

Merging Categories [page 270]
When you create a new model using categories, you can see the existing categories, as well as the three standard Streaming categories. You can merge existing model categories with the Streaming category.

Changing the Default Category [page 270]
Change the default category to the Streaming category, so that you can create new Streaming models.

11.3.2.1 Merging Categories

When you create a new model using categories, you can see the existing categories, as well as the three standard Streaming categories. You can merge existing model categories with the Streaming category.

Procedure

1. Choose **Tools** > **Resources** > **Model Category Sets**.
2. From the list in the dialog, select the set you want to add to the Streaming category.
3. Click the **Merge** button in the toolbar.
4. Select **Streaming** from the list and choose **OK**.

11.3.2.2 Changing the Default Category

Change the default category to the Streaming category, so that you can create new Streaming models.

Procedure

1. From the PowerDesigner main menu, choose **Tools** > **General Options**.
2. Under Category, select **Model Creation**.
3. In the Model Creation frame, with **Enable categories** checked, select a default category set.
4. Click **OK**.

11.3.3 Setting Datatypes for a Streaming Schema

Manually set the datatype attribute for a Streaming Schema definition if the Streaming Datatype column in the Attributes tab of a Streaming Schema definition is empty or shows the wrong values.

Context

You may need to set datatypes for a logical data model you generate from a physical data model, if the generation process cannot determine how to convert the database datatype to a Streaming datatype. Datatypes for the shipped sample model are set correctly and no further adjustments are necessary.

Procedure

1. Right-click a schema definition and choose **Properties**.
2. Click the **Attributes** tab and review values in the Streaming Datatype column.

   For example, in the sample model, the Bond Quote Attributes shows the following datatypes:

Attribute Name	Streaming Datatype
Instrument	`string`
Quote Date	`seconddate`
Quote Sequence Number	`integer`
Quote Time	`msdate`
Ask Price	`money(4)`
Ask Size	`integer`
Bid Price	`money(4)`
Bid Size	`integer`
Yield	`money(2)`

   If values are missing or incorrect, continue with steps 3 - 5.

3. Click **Customize Columns and Filter**  (**Ctrl+U**).
4. If needed, adjust columns available for viewing:
   a. Unselect **Data Type**, **Length**, and **Precision**.
   b. Select:

      Name
      Streaming Datatype
      Length
      Precision
      Mandatory
      Primary Identifier
      Displayed (selected by default)
5. Use the controls below the list to adjust the order so that **Primary Identifier** and **Displayed** are the last two checkboxes.

Results

Performing this task once corrects the datatypes for all schema definitions.

11.4 Streaming Schema Model Development

Develop schema using the PowerDesigner extensions.

You can:

- Explore the sample model
- Create a schema model using categories, or by creating and extending a logical data model
- Add schema to models
- Validate your schema with built-in checks, as well as custom ones
- Import defined schema definitions into a Streaming Schema model from CCL files
- Export schema definitions from the Streaming Schema model into CCL files

In this section:

Exploring the Sample Model [page 272]
Review the sample model from the sample project.

Creating a Streaming Schema Model [page 273]
Create a new Streaming Schema model using the Streaming Schema category, either by creating a logical model and extending it, or by generating it from an SAP IQ, SAP HANA, or SAP ASE model that has been extended.

Validating a Model [page 280]
Check the validity of your model after schema changes, and before generating schema templates, code, or a physical model. You can check the validity of a model at any time.

Importing a CCL File [page 282]
Import the defined schema definitions in a CCL file into a Streaming Schema model.

Exporting a CCL File [page 283]
Export all the defined schema from the Streaming Schema model into a CCL file for compiling and further analysis.

11.4.1 Exploring the Sample Model

Review the sample model from the sample project.

Prerequisites

Install the sample model and complete the extended model setup.

Procedure

1. Start PowerDesigner and open the sample project with the sample model.
2. To open any of the models, either:
    - Double-click the model in the Model Relationship Diagram, or,
    - In the Browser tree, double-click the model, or right-click and choose **Open** or **Open as read-only**.

    > Note
    >
    > Do not save changes to the installed sample model. Save it to another folder so that a new version of the model and project are created.

3. To display the sample schema definitions in the Streaming Schema model, expand the navigation buttons in the Browser tree.
4. To see more information on a schema definition:
    - Right-click the schema definition, an identifier, or an attribute in the tree view and choose **Properties**, or,
    - Right-click the schema definition in the tree view and choose **Find in Diagram**.

    Explore the SAP IQ, SAP HANA, and SAP ASE models in the same way.

11.4.2 Creating a Streaming Schema Model

Create a new Streaming Schema model using the Streaming Schema category, either by creating a logical model and extending it, or by generating it from an SAP IQ, SAP HANA, or SAP ASE model that has been extended.

In this section:

11.4.2.1 Creating a Model Using Categories

Use PowerDesigner to create and automatically extend any Streaming Schema model type.

Prerequisites

Designate the Streaming Schema set as the default category.

Procedure

1. Choose **File** > **New Model**.
2. In the New Model dialog, select **Categories**, and choose a category item:
   - **Streaming Schema**
   - **SAP IQ**
   - **ASE**
   - **HANA**
3. Enter a model name.
4. Click **OK**.

11.4.2.2 Creating a Logical Data Model

Create a logical data model and add extensions to it.

Procedure

1. Choose **File** > **New Model**.
2. In the New Model dialog, select **Model types** and **Logical Data Model**.
3. Enter a model name.
4. Click the **Select Extensions** button  to the right of the Extensions box.

   A dialog shows currently loaded extensions. You can apply extensions when you create the model or later.
5. Select **Streaming Schema**, then select whether to share or copy:

Option	Description
Share the extended model definitions	PowerDesigner always uses the contents of the `.xem` file. If the contents of the `.xem` file change, the model sees those changes. For example, if a future version of Streaming Schema includes a new version of the file, models that share it sees those changes immediately.
Copy the extended model definitions	Copies the contents of the `.xem` file into the model. The model uses its local copy instead of the file on disk.

6. Click **OK**

   With either approach, use other extensions besides the shipped Streaming Schema extensions by creating your own `.xem` file. Although it is possible to do this by adding to the `StreamingSchema.xem` file, SAP does not recommend this.

11.4.2.3 Adding Schema Definition

Add a schema definition by creating it, importing schema definitions in a CCL file, or generating it from an SAP IQ, SAP HANA, or SAP ASE table.

In this section:

Creating Schema from the Schema Definitions Container [page 275]
Create a new schema definition with initial properties.

Creating Schema with the Entity Tool [page 276]
Create schema from the diagram.

Creating a Schema from the Streaming Schema Container [page 276]
Create a new schema definition with initial properties.

Generating Schema from an SAP IQ, SAP HANA, or SAP ASE Table [page 276]
Follow the same steps as when generating a Streaming Schema model, selecting a single table to generate.

11.4.2.3.1 Creating Schema from the Schema Definitions Container

Create a new schema definition with initial properties.

Procedure

1. Open the PowerDesigner model.
2. In the Browser tree, right-click the Streaming Schema container and choose **New**.
3. Complete the information in the **General** tab or other tabs.

   You can complete schema definition properties at any time before generating the physical models.

4. Click **OK** to save the schema definition.

11.4.2.3.2 Creating Schema with the Entity Tool

Create schema from the diagram.

Procedure

1. Open the PowerDesigner model.
2. In the diagram, click the **Entity** tool .

   A new, empty schema definition appears in the diagram, and in the Browser tree when expanded.
3. Right-click the diagram and choose **Properties**.
4. Add attributes and identifiers in the properties sheet.

11.4.2.3.3 Creating a Schema from the Streaming Schema Container

Create a new schema definition with initial properties.

Procedure

1. Right-click the Streaming Schema container and choose **New** > **Streaming Schema**.
2. Complete the information in the **General** tab or other tabs.

   You can complete schema definition properties at any time before generating the physical models.
3. Click **OK** to save the schema definition.

11.4.2.3.4 Generating Schema from an SAP IQ, SAP HANA, or SAP ASE Table

Follow the same steps as when generating a Streaming Schema model, selecting a single table to generate.

11.4.2.4 Defining Schema Properties

Define schema details in the properties sheet.

Prerequisites

Add the schema definition to the Streaming Schema model.

Procedure

1. Open the Streaming Schema Properties sheet from the Browser tree or the diagram.
2. Edit fields on the **General**, **Attributes**, and **Identifiers** tabs.
3. (Optional) Right-click an attribute to open the Attribute Properties sheet.
4. (Optional) In the Attribute Properties sheet, choose **More** to see extended property details.
5. Click **Apply** to apply changes.
6. Click **OK** when done.

In this section:

General Tab Properties [page 278]
View information about the Name and Comment properties of a schema definition on the General tab of the Schema Definition Properties sheet.

Attributes Tab Properties [page 278]
The Attributes tab of the Schema Definition Properties sheet lets you quickly set information for all fields of a Streaming Schema.

Adding an Attribute to Schema [page 279]
Add fields to schema by adding attributes to the schema definition.

Identifiers [page 279]
An identifier is a column or combination of columns that uniquely defines a specific Streaming Schema.

Defining Identifiers [page 280]
Define identifiers to indicate which schema attributes become keys in the destination table.

11.4.2.4.1 General Tab Properties

View information about the Name and Comment properties of a schema definition on the General tab of the Schema Definition Properties sheet.

Property	Description
Name	Text that identifies the object's purpose for non-technical users, for example, Stock Quote. This element is used for descriptive purposes only, and can contain any string.
Comment	An optional comment field. This is stored only in the model, not in the schema.

11.4.2.4.2 Attributes Tab Properties

The Attributes tab of the Schema Definition Properties sheet lets you quickly set information for all fields of a Streaming Schema.

Property	Description
Name	Name of the field. The value can contain any string. This element is used for descriptive purposes only.
Code	By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the Code field.
Smart data streaming Datatype	Select from the list of supported smart data streaming datatypes. For information on smart data streaming datatypes, see Datatypes [page 95].
Data Type (Internal PowerDesigner datatype)	Select from the list of supported datatypes. For information on PowerDesigner datatypes, see related information in *PowerDesigner>Data Modeling*.
Length	Required for money and decimal data. Limited to precision 34. Precision must be the same on SAP IQ and SAP ASE. Not used for other datatypes.
Precision	Required for money and decimal data. Not used for other datatypes.

Property	Description
Domain	Specifies a domain which defines the datatype and related data characteristics for the schema attribute. It may also indicate check parameters, and business rules. Select a domain from the list, or click the Ellipsis button to create a new domain in the List of Domains.

Attribute Properties Sheet

Each field in a schema definition has its own Properties sheet.

In the Attribute Properties Sheet, you can:

- View or edit the same information as in the Attributes tab of the Schema Definition Properties sheet
- Specify validation checks for an attribute
- View attribute dependencies
- View impact and lineage analyses for an attribute

11.4.2.4.3 Adding an Attribute to Schema

Add fields to schema by adding attributes to the schema definition.

Procedure

1. In the schema definition to which you are adding an attribute, do any of the following:
    - From the schema definition, right-click and choose **New**. This opens the Attribute Properties sheet.
    - From the Attributes tab in the Streaming SchemaProperties sheet, type information in the row below the last attribute.
    - From the Attributes tab in the Streaming Schema Properties sheet, click the **Insert a Row**, **Add a Row**, or **Add Attributes** or **Replicate Attributes** toolbar button from other schema definitions.

    Before replicating attributes, read *Object Replications* in the *PowerDesigner Core Features Guide*.
2. Edit information in the Attributes Properties sheet or row as needed.

11.4.2.4.4 Identifiers

An identifier is a column or combination of columns that uniquely defines a specific Streaming Schema.

Identifiers in the Streaming Schema model become keys on tables in the SAP IQ, SAP HANA, and SAP ASE physical models.

Each Streaming Schema can have at most one primary identifier, which becomes the primary key in the generated table.

When an identifier has multiple attributes, the primary key in the destination table is composed of multiple columns. For example, in the sample model, the Dividend Event schema has one identifier. Attributes for this primary identifier are Instrument and Disbursed Date. Thus the primary key for the Dividend Event table is composed of both the Instrument and Disbursed Date columns.

11.4.2.4.5 Defining Identifiers

Define identifiers to indicate which schema attributes become keys in the destination table.

Procedure

1. Choose one of the following:
   - Right-click Streaming Schema and choose **New** > **Identifier**.
   - (Primary identifiers only) On the Streaming Schema Properties sheet, select the **Attributes** tab, and click the Primary Identifier column (the narrow column with the header **P**) for each attribute that is part of the primary identifier. Skip the remaining steps.

   > Note
   >
   > In the Streaming Schema Properties **Attributes** tab, a checkmark in the **P** column indicates a primary identifier.
2. Select the **General** tab in the Identifier Properties sheet:
   a. (Optional) Set the identifier name.
   b. For a primary key, select **Primary Identifier**.
3. On the **Attributes** tab in the Identifier Properties sheet, enter the fields that identify the schema.

11.4.3 Validating a Model

Check the validity of your model after schema changes, and before generating schema templates, code, or a physical model. You can check the validity of a model at any time.

Procedure

1. (Optional) Select diagrams for the schema you want to validate.
2. Choose **Tools** > **Check Model (F4)**.

3. In the **Options** tab of Check Model Parameters, expand the containers and choose validation checks.

   The **Options** tab lists checks to be performed with symbols indicating their severity.
   - Do not disable any smart data streaming-specific checks.
   - (Default and recommended) Disable **Existence of relationship or entity link** under **Entity**.
4. In the **Selection** tab, navigate to the Streaming Schema subtab and select schema definitions to check:
   - Select or unselect check boxes.
   - Choose a named selection.
   - If you selected schema in your diagram before starting the model check, you can select them for checking by clicking **Use Graphical Selection** ( ) in the Selection tab toolbar.
5. Click **OK**.

Next Steps

Review results in the **Check Model** subtab in the status bar. It lists the checks made, and any errors or warnings.

Correct any errors. No automatic corrections are provided.

In this section:

11.4.3.1 PowerDesigner Validity Checks

Standard PowerDesigner checks determine if a model is internally consistent and correct.

For example:

- Each Streaming Schema name must be unique
- Each object name in the Streaming Schema model must be unique
- Each field must have an assigned smart data streaming datatype.

For descriptions of standard PowerDesigner checks, see *Working with Data Models* > *Checking a Data Model* in the PowerDesigner *Data Modeling* guide.

11.4.3.2 Custom Checks for Streaming Schema Extensions

The Streaming Schema extension offers many custom checks.

Checks for Each Schema

Custom checks under the Schema Definition type validate values in the **General** tab of the Schema Properties sheet.

Option	Description
NameIsValid	Validates that names of Streaming Schema are valid java identifiers.

Checks for Each Field in a Schema

Custom checks for fields are under the Entity Attribute type. They validate values in the Attributes tab of the Streaming Schema Properties sheet.

Option	Description
FieldNameIsValid	Validates that field names are valid java identifiers.
StreamingDatatypeExists	Validates that the datatype is specified.
UniqueDestColumnName	Validates that DestColumnName is unique within that schema.

11.4.4 Importing a CCL File

Import the defined schema definitions in a CCL file into a Streaming Schema model.

Prerequisites

You have added the Everyone user to the PowerDesigner folder and have granted this user read and write permissions.

Procedure

1. Open the PowerDesigner model.

2. In the **Browser** tree, right-click the Streaming Schema container and choose **Import CCL File...**.
3. Navigate to the CCL file you wish to import.
4. Click **OK** to import the schema definitions defined in the CCL file.

> **Note**
>
> A warning message appears if the CCL file schema definitions are not valid. Resolve the errors before importing the CCL file. Navigate to the `User/Appdata/Roaming/PowerDesigner/StreamingCompiler/compiledOutput.log` file to view the errors.

Results

The schema defined in the CCL file is imported into the Streaming Schema model.

11.4.5 Exporting a CCL File

Export all the defined schema from the Streaming Schema model into a CCL file for compiling and further analysis.

Procedure

1. Open the PowerDesigner model.
2. In the **Browser** tree, right-click the Streaming Schema container and choose **Export CCL File...**.
3. Navigate to the CCL file you wish to export to.
4. Click **OK** to export the schema definitions.

Results

The schema defined in the Streaming Schema model is exported as a CCL file.

11.5 Model Generation

Model generation with the Streaming Schema models is a critical step in ensuring the integrity of your production environment.

To generate data models, you can perform one of the following tasks:

- Generate SAP IQ, HANA, and ASE physical data models from the Streaming Schema model.
- Generate a Streaming Schema logical data model from the SAP IQ, HANA, or ASE physical data model.

Each generation process relies on transformation rules for that model type, which are defined in the Streaming Schema extensions for PowerDesigner.

In this section:

11.5.1 Generating a new SAP IQ, HANA, or ASE Model from a Streaming Schema Model

Generate either an SAP IQ, HANA, or ASE physical data model from a Streaming Schema logical data model.

Procedure

1. Open the Streaming Schema model.
2. From the PowerDesigner main menu, choose **Tools** > **Generate Physical Data Model**.
3. In the **General** tab of the PDM Generation Options dialog, choose **Generate new Physical Data Model**.
4. For a new model, choose the target DBMS and the appropriate Name and Code.
   - For HANA, choose:

Field	Value
DBMS	**SAP HANA Database 1.0**
Name	Keep the default, StreamingSchema, or enter another name.
Code	Auto-generated from Name. For example, when Name is StreamingSchema, Code is StreamingSchema.

   - For ASE, choose:

Field	Value
DBMS	**SAP Adaptive Server Enterprise 15.7**
Name	Keep the default, StreamingSchema_1 (the name of the container), or enter another name.
Code	Auto-generated from Name. For example, when Name is StreamingSchema_1, Code is StreamingSchema_1.

- For IQ, choose:

Field	Value
DBMS	**SAP IQ 15.X** Note Use latest version of SAP IQ available.
Name	Keep the default, StreamingSchema_1 (the name of the container), or enter another name.
Code	Auto-generated from Name. For example, when Name is StreamingSchema_1, Code is StreamingSchema_1.

5. Click the **Detail** tab.
6. (Optional) Choose **Check model** and **Save generation dependencies**.
7. Ensure that **Enable transformations** is selected.
8. Click the **Extensions** tab and ensure that the appropriate extension is selected:
   - **ASE** when generating a new SAP ASE model
   - **IQ** when generating a new SAP IQ model
   - **HANA** when generating a new SAP HANA model
9. On the **Pre-generation** and **Post-generation** tabs, ensure that all transformation rules are selected.
   The post-generation tab appears only for new models.
10. On the **Selection** tab, select **StreamingSchema** to create tables for SAP IQ, SAP HANA, or SAP ASE, and choose **OK**.

Next Steps

After generation, check indexes, set physical options, and add foreign keys as needed.

In this section:

11.5.1.1 Checking Indexes

PowerDesigner creates default indexes. Add, edit, or remove them as needed.

Procedure

1. Open the new or updated physical data model.
2. For each table, right-click the table and choose **Properties**.
3. In the **Indexes** tab, edit indexes as needed for your data and performance requirements.

11.5.1.2 Setting Physical Options

Set physical options for each table as needed for your SAP IQ or SAP ASE database.

Procedure

1. Right-click the table and choose **Properties**.
2. Define any options needed:
    - (SAP ASE only) In the **Physical Options (Common)** tab, choose from the physical options most commonly set for the object.
    - In the **Physical Options** tab, choose from all available options.
    - (SAP ASE only) In the **Partitions** tab, set partitioning options for selected columns.

    For more information on partitioning, see the SAP ASE and SAP IQ documentation sets.

11.5.1.3 Adding Foreign Keys

Add foreign-key relationships to physical data models.

Procedure

1. Add tables to the physical data model that are not in your Market Data diagram and that contain lookup columns for foreign keys.

    New SAP ASE, SAP HANA, and SAP IQ models generated from a Streaming Schema model contain only market data tables.

2. Right-click the table and choose **Properties** or **Keys**.
3. Add foreign-key relationships to tables that are not in the Market Data diagram.

11.5.2 Generating a New Streaming Schema Model from an SAP IQ, HANA, or ASE Model

Generate a new Streaming Schema logical data model from either SAP IQ, HANA, or ASE physical data models.

Procedure

1. Open the **IQ**, **HANA**, or **ASE** model.
2. From the PowerDesigner main menu, choose **Tools** > **Generate Logical Data Model**.
3. In the **General** tab of the LDM Generation Options dialog, choose **Generate new Logical Data Model**.
4. Specify a **Name**.
   Code is autogenerated from the name.
5. On the **Detail** tab, choose Options:
   - (Optional) Check model
   - (Optional) Save generation dependencies
   - (Optional) Convert names into codes
   - (Required) Enable transformations
6. On the **Extensions** tab, choose **StreamingSchema**.
7. On the **Selection** tab, choose tables from which to generate schema.
8. Click **OK**.

11.5.3 Updating an Existing SAP IQ, HANA, or ASE Model from a Streaming Schema Model

Update either an SAP IQ, HANA, or ASE physical data model from a Streaming Schema logical data model.

Procedure

1. Open the Streaming Schema model.
2. From the PowerDesigner main menu, choose **Tools** > **Generate Physical Data Model**.
3. In the **General** tab of the PDM Generation Options dialog, choose **Update existing Physical Data Model**.

4. Select the model and leave **Preserve Modifications** selected.
5. Click the **Detail** tab.
6. (Optional) Choose **Check model** and **Save generation dependencies**.
7. Ensure that **Enable transformations** is selected.
8. In the Merge Models dialog, confirm the updates you want, and click **OK**.

Next Steps

After generation, check indexes, set physical options, and add foreign keys as needed.

11.5.4 Updating an Existing Streaming Schema Model from an SAP IQ, HANA, or ASE Model

Update an existing Streaming Schema logical data model from either SAP IQ, HANA, or ASE physical data models.

Procedure

1. Open either the **IQ**, **HANA**, or **ASE** model.
2. From the PowerDesigner main menu, choose **Tools** > **Generate Logical Data Model**.
3. In the **General** tab of the LDM Generation Options dialog, choose **Update existing Logical Data Model**.
4. Select the model and leave **Preserve Modifications** selected.
5. On the **Detail** tab, choose Options:
    - (Optional) Check model
    - (Optional) Save generation dependencies
    - (Optional) Convert names into codes
    - (Required) Enable transformations
6. On the **Selection** tab, choose tables from which to generate schema.
7. Click **OK**.

11.6 Impact and Lineage Analysis

With impact and lineage analysis, you can determine the full impact of changes to any object in the integrated model.

Impact analysis shows the effect of an action on the objects that depend on the initial object.

Lineage analysis identifies the objects that influence the initial object.

You can perform these analyses on:

- A schema definition or any of its properties in the Streaming Schema logical data model
- A table or column in the SAP ASE, HANA, or IQ physical data model

The results shows the effect of a change throughout the logical and physical data models.

In this section:

11.6.1 Launching an Impact and Lineage Analysis

Analyze the impact of a change to your model from the Impact and Lineage Analysis dialog box.

Context

The Impact and Lineage Analysis dialog lets you review your analysis through:

- A preview – displays the impact and lineage analysis in a tree form. See *PowerDesigner Core Features Guide* > *Reviewing an Analysis in Preview*.
- An impact analysis model (IAM) – displays the impact and lineage analysis in a diagram. See *PowerDesigner Core Features Guide* > *Reviewing an Analysis in an IAM Model*.

Procedure

1. Open an impact and lineage analysis in any of these ways:
   - Select an object in the Browser or in the diagram and press **Ctrl** + **F11**.
   - Select one or more objects in the diagram and select **Tools** > **Impact and Lineage Analysis**.

- Right-click an object symbol in the diagram and select **Edit** > **Impact and Lineage Analysis**.
- Right-click an object entry in the Browser and select **Impact and Lineage Analysis**.
- (When deleting an object) Click **Impact** on the Confirm Deletion dialog box.
- Open an object's property sheet, click the **Dependencies** tab, then click **Impact Analysis** .

2. (Optional) Enter a name for your analysis result. This becomes the name of the generated model.
3. Select an impact rule set for your analysis. Choose one of these predefined rule sets:
   - Conceptual Impact Analysis – restrict the analysis to objects impacted by modeling changes on the initial object, such as a modification on a requirement definition.
   - Data Impact Analysis – identify the use, if any, of a value contained in the initial object.
   - Delete Impact Analysis – (default when deleting an object) restrict the analysis to objects that are directly impacted by the deletion of the initial object.
   - Global Impact Analysis – (default when not deleting an object) identify all the objects that depend on the initial object.
   - None – no impact rule set is selected.
4. Select a lineage rule set for your analysis. Choose one of these predefined rule sets:
   - Conceptual Lineage Analysis – justify the modeling existence of the initial object, and ensure it fulfills a well-identified need.
   - Data Lineage Analysis – identify the origin of the value contained in the initial object.
   - Global Lineage Analysis – (default when not deleting an object) identify all the objects that influence the initial object.
   - None – (default when deleting an object) no lineage rule set is selected.
5. (Optional) Click the **Properties** tool next to each rule set to review it. See *PowerDesigner Core Features Guide* > *Editing analysis rules*.

Results

The analysis appears in the **Impact and Lineage** tab of the dialog box. See *PowerDesigner Core Features Guide* > *Reviewing an Analysis in Preview*.

Note

You can click the **Select Path** tool to change the default folder for analysis rule sets, or click the **List of Rule Sets** tool to open the **List of Impact and Lineage Analysis Rule Sets** window, and review a specific rule.

11.6.2 Generating an Analysis Diagram

Generate an analysis diagram to view the impact or lineage analysis in graphical form.

Prerequisites

Launch an impact or lineage analysis.

Procedure

1. In the Impact and Lineage Analysis dialog, click **Generate Diagram** to view a graphical form of the analysis in its default diagram.
2. (Optional) Save (**Ctrl+S**) the diagram as an impact analysis model (IAM).
   See *PowerDesigner Core Features Guide > Reviewing an Analysis in an IAM Model*.

11.6.3 Reviewing an Impact and Lineage Analysis

Review the analysis in the preview or the impact and lineage model diagram.

Procedure

1. Review the impact of the action and the lineage of the entity in the preview.
2. In the preview **List** tab, save the analysis in RTF or CSV format, or print.
3. You can refine your analysis by removing or adding initial objects, changing the analysis rule sets to be used, and customizing actions.
4. If you have generated an IAM, you can customize the display preferences and model options, print the model, and compare it with another IAM.
5. Watch for a red dot on an object icon in a generated model.

   When you generate a model to another model or create an external shortcut, you create cross-model dependencies, which are taken into account during impact and lineage analysis.

   When an object belonging to an unavailable related model is encountered, a red dot appears on the object icon and the analysis is interrupted. To continue, open the related model by right-clicking the object in the IAM Browser or in the preview, and selecting **Open Model**.

11.6.4 Sample Analysis for a Schema Definition

The sample analysis for a schema definition shows that the Bond History schema in the Streaming Schema model was used to generate the `BOND_HISTORY` tables in the SAP HANA, ASE, and IQ models.

11.6.5 Sample Analysis for a Table

The sample analysis for a table shows that the STOCK_QUOTE table was generated from the Stock Quote schema definition in the Streaming Schema model.

Outgoing References shows foreign-key relationships. Streaming Schema definitions become Market Data diagram tables when generated to a PDM.

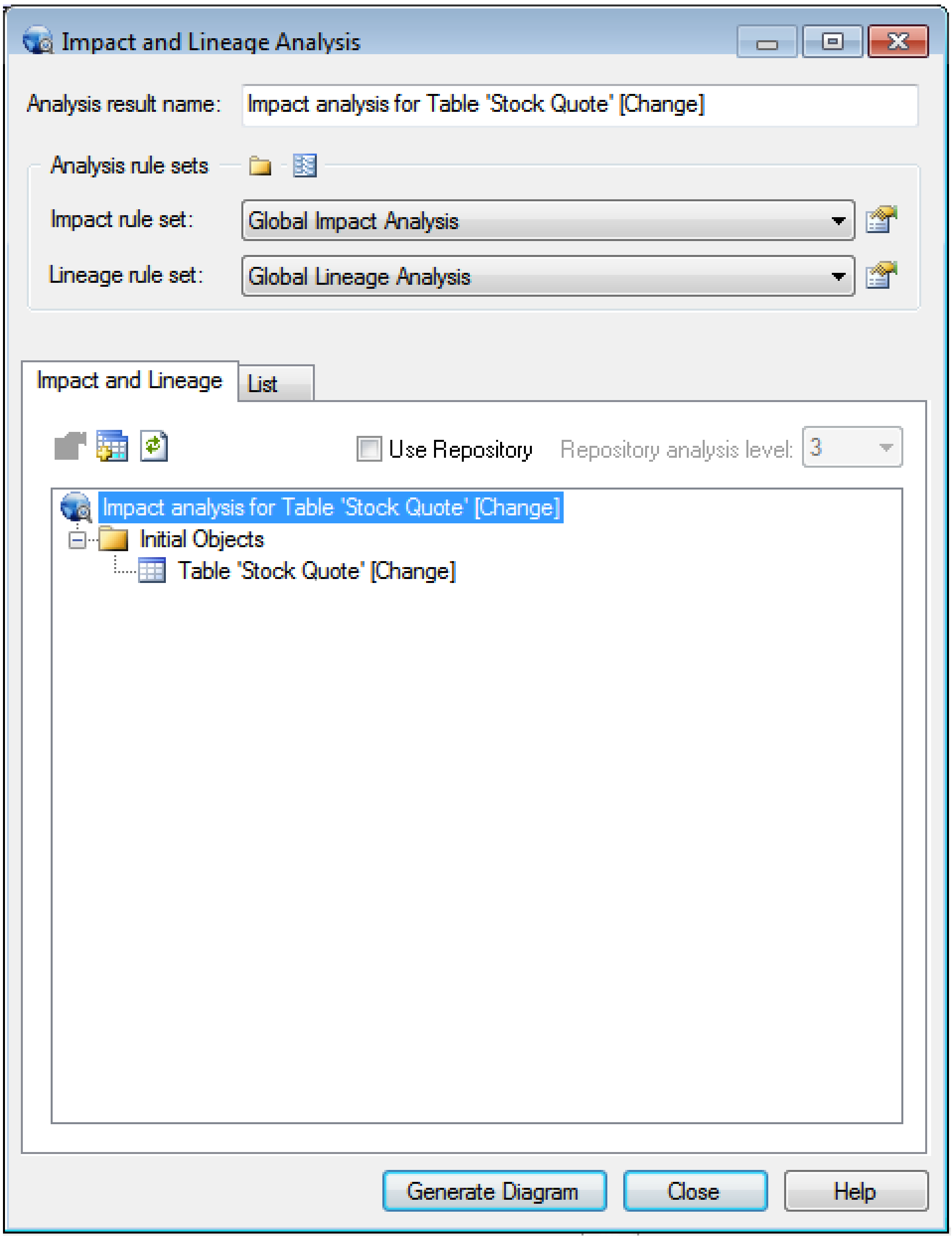


Figure 6: Impact and Lineage Analysis for STOCK_QUOTE Table in ASE

11.7 DDL Script Generation

The data models for the SAP IQ, HANA, and ASE databases target different databases; however, they share an almost identical structure. Modify data models by creating additional tables or columns to suit your business environment.

The corresponding adapter (SAP IQ, HANA, ASE) schema must match the SAP IQ, HANA, and ASE database schema for all tables in which data is inserted. After you make changes, use PowerDesigner to produce a set of data definition language (DDL) statements directly from the physical data model (SAP IQ, HANA, and ASE). PowerDesigner saves the DDL statements in a SQL script to generate the tables and other objects for the target databases.

In this section:

11.7.1 Generating Database Schema with PowerDesigner

PowerDesigner includes all of the necessary resources to generate a set of DDL statements in SQL scripts directly from the PowerDesigner data models. Run these scripts to generate a schema for your SAP IQ, HANA, and ASE databases.

Procedure

1. In PowerDesigner, open the data model.
2. Change the default database user.
3. Generate the script that creates a schema for the new database.
4. Log in to the database and run the script.

In this section:

11.7.1.1 Changing the Default Database User

Overwrite the default database owner for the SAP IQ, HANA, or ASE database with a name specific to your environment.

Context

In the database, the user who creates an object (table, view, stored procedure, and so on) owns that object and is automatically granted all permissions on it. Overwriting the default user name globally changes ownership of database objects from the default owner to the new owner.

Procedure

1. Start PowerDesigner.
2. Select **File** > **Open** and choose the database that you want to change the default owner of (`IQ.pdm`, `HANA.pdm`, or `ASE.pdm`).
3. Select **Model** > **Users and Roles** > **Users** .
4. In the Name and Code columns, change the default user to the new database user.
5. Click **OK**.

11.7.2 Generating DDL Scripts

Generate DDL scripts directly from the SAP IQ, HANA, or ASE data model. PowerDesigner saves the results in a SQL script that you can use to generate the tables and other objects in the target database.

Context

Use the model file of the database for which you wish to generate DDL scripts. For example, to generate DDL for the SAP ASE database, use the ASE model. When you have the model open, do not change the target database as doing so results in the loss of index information.

By default, the `ASE.pdm` data model includes only those indexes that support the sample queries. The statements that create these indexes are included in the DDL scripts, which means the indexes supplied with the model are created automatically when you run the corresponding DDL scripts.

You can add or remove indexes from the ASE data model. See the SAP ASE product documentation for details on SAP ASE indexes.

Procedure

1. Select **Database** > **Generate Database**.
2. Browse to the directory where you want to store the script. Click **OK**.
3. Enter a name for the SQL script.
4. On the **Options** tab, verify that the options are set correctly:

Object	Options
Domain	Create User-Defined Data Type
Table	Create Table
Column	User Data Type
Key	Create Primary Key Inside
Index	○ Create Index ○ Index Filter Foreign Key ○ Index Filter Alternate Key ○ Index Filter Cluster ○ Index Filter Others

5. Click the **Selection** tab.
6. Choose the database owner.
7. On the **Tables** tab, click **Select All**.
8. On the **Domains** tab, choose the database owner, click **Select All**, click **Apply**, then click **OK**.

   PowerDesigner checks the model for any errors, builds a result list, and generates the DDL. The Result dialog appears, which identifies the name and location of the generated file.
9. (Optional) Click **Edit** to view the generated script.

   The Result List dialog appears in the background and may include several warnings, for example, `"Existence of index"` and `"Existence of reference".` You can safely ignore these warnings.
10. Close the Result List dialog, then exit PowerDesigner.
11. PowerDesigner prompts:
    - To save the current workspace, click **No**.
    - To save the (modified) model, click **Yes** or **No**.

Results

- PowerDesigner prompts: To save the current workspace, click **No**.
- If PowerDesigner prompts you to save the model, click **Yes** to save the modified model. Otherwise, click **No**.

11.7.3 Executing DDL Scripts for the SAP IQ Database

Execute the DDL script in Interactive SQL and create database objects in the SAP IQ database.

Prerequisites

Start the SAP IQ database server if it is not running.

Procedure

1. In a command prompt, change to the directory that contains the database files and enter:

   ```
   start_iq -n <server-name> @<config-file>.cfg <database-name>.db.
   ```

   Use the `-n` switch to name the server, either in the configuration file or on the command line when you start the server.

   > **Note**
   >
   > If you specify `-n` `<server-name>` without a `<database-name>`, you connect to the default database on the current server. If you specify `-n` `<database-name>` without a `<server-name>`, you connect to the specified database on the current server.

2. Enter `dbisql`.
3. Enter the correct user ID, password, and server information.
4. Open the generated DDL script for SAP IQ and click **Execute SQL Statement** on the toolbar.

11.7.4 Executing DDL Scripts for the SAP HANA Database

Execute the DDL script using hdbsql and create database objects in the SAP HANA database.

Prerequisites

Start the SAP HANA database server if it is not running.

Procedure

- In a command prompt, enter:

```
hdbsql -n <host>:<port> -u <user> -p <password> -I <script-file>
```

11.7.5 Executing DDL Scripts for the SAP ASE Database

Execute the DDL script in Interactive SQL and create database objects in the SAP ASE database.

Prerequisites

Start the SAP ASE server if it is not running.

Procedure

1. In a command prompt, enter:

```
isql -S <server-name> -U <user-name> -P <password> -i <ase-ddl.sql-file> -o
<log-file>
```

2. PowerDesigner prompts:
   - To save the current workspace. Click **No**
   - To save the (modified) model. Click **Yes** or **No**.
3. Check the log file for errors.

12 Appendix: Tutorial for Building and Testing a Project in Studio

Walk through this hands-on tutorial to create, compile, and test a simple project in the studio. This is a great way of getting started in and familiar with studio and SAP HANA smart data streaming.

Context

The portfolio valuation sample project demonstrates how you can easily define streams and windows by attaching a previously configured adapter and discovering its schema, or by manually defining a window and its schema. It shows you how to define continuous queries—aggregations, joins, and more—using the visual tools in the SAP HANA Streaming Development perspective.

After you build the project, the tutorial shows you how to compile, run, and test the project you have created using tools in the SAP HANA Streaming Run-Test perspective.

1. About the Portfolio Valuation Sample Project [page 300]
   The portfolio valuation project that you build in this tutorial applies current prices to a portfolio of investments to compute the value of each investment and of the portfolio. It uses simple queries to aggregate and join data from two input windows, then outputs the refined data into SAP HANA using an output adapter.
2. Creating the Project [page 304]
   In studio, define a new set of processing instructions for event data.
3. Editing a Project Diagram [page 305]
   Edit projects in the visual editor by adding shapes from the palette to the project diagram, connecting them, and completing the configuration of each shape.
4. Adding an Input Adapter [page 306]
   Attach an adapter by inserting it in the diagram, connecting it to a stream or window, and setting properties.
5. Discovering a Schema [page 308]
   In studio, use the **Schema Discovery** button in the adapter shape to discover and automatically create a schema based on the format of the data from the adapter.
6. Adding an Input Window Manually [page 310]
   In studio, add an input window to the diagram in the sample project `my_portfolio_valuation`.
7. Creating an Aggregate as a Simple Query [page 311]
   In studio, add an aggregate simple query to the sample diagram to create a volume weighted average price (VWAP).
8. Creating a Join as a Simple Query [page 313]
   Add a join to the sample project. A join combines events from two or more inputs to create a single stream or window. It is similar to a join in SQL.
9. Preparing Data for Output [page 317]
   Add a new aggregate to finalize the data and prepare it for output into SAP HANA.

10. Adding an Output Adapter for SAP HANA [page 318]
    Create an SAP HANA service definition, and insert and configure an SAP HANA Output adapter using the studio.
11. Preparing to Compile [page 320]
    Clean up the diagram by removing unused elements.
12. Compiling the Project and Viewing Problems [page 321]
    Compile a project in studio before running it to check for errors and make corrections. Use the Problems view to view error details.
13. Deploying the Sample Project [page 323]
    Run the project and watch it open in the SAP HANA Streaming Run-Test perspective.
14. Loading Data into the Sample Project [page 323]
    Test the sample project by loading reference data into the Positions window.
15. Testing the Project with Recorded Data [page 324]
    Play back the previously recorded price feed data, and view the continuous portfolio valuations in the sample project.
16. Sample Project with Modules [page 326]
    This variation of the portfolio valuation project provides a complex, real-world example. The project uses a defined module with a named schema to easily scale out the application in a very high volume deployment. The SAP HANA Output adapter is not included in this module.

12.1 About the Portfolio Valuation Sample Project

The portfolio valuation project that you build in this tutorial applies current prices to a portfolio of investments to compute the value of each investment and of the portfolio. It uses simple queries to aggregate and join data from two input windows, then outputs the refined data into SAP HANA using an output adapter.

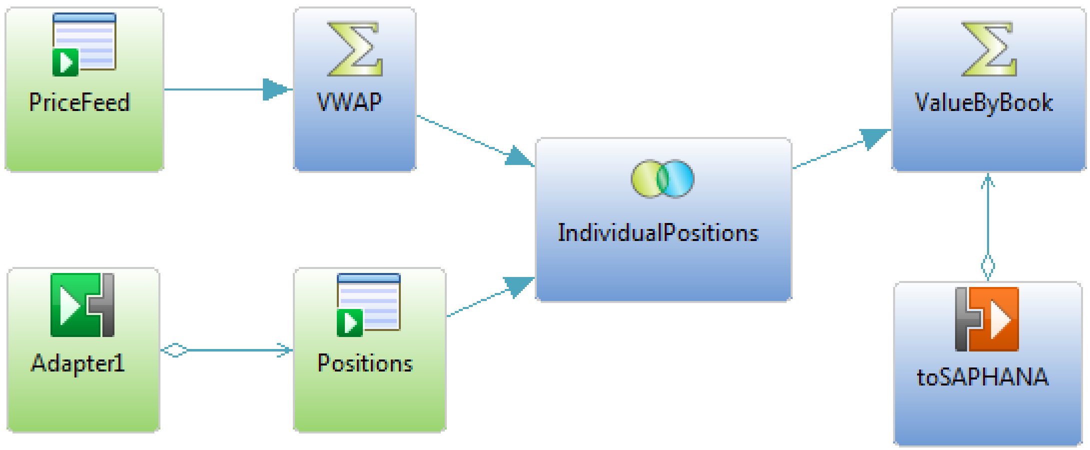
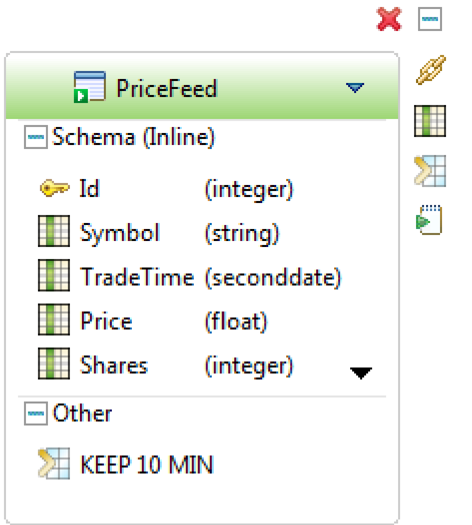
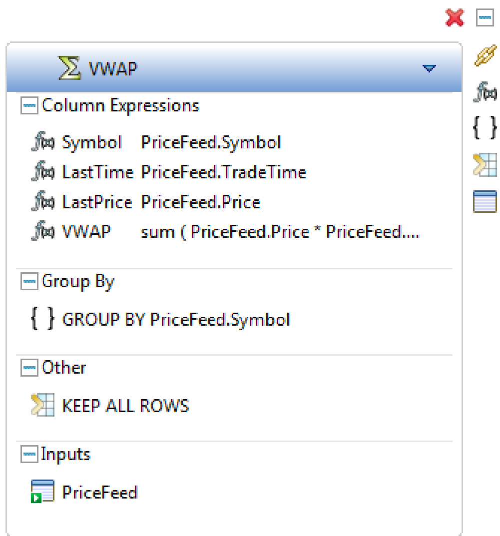


Figure 7: Portfolio Valuation Sample Diagram (Iconic Mode)

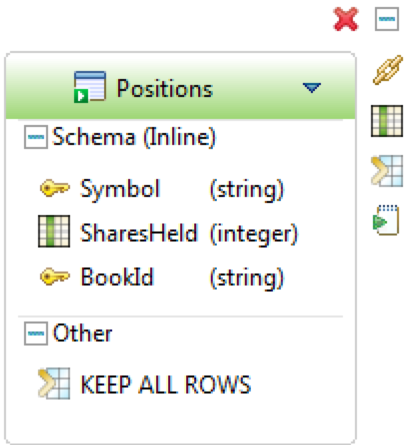
The portfolio valuation project consists of various elements, any of which you can create either in the visual editor, or by manually writing CCL code.

<table>
<tr><th>CCL Element</th><th>Diagram Element</th></tr>
<tr><td>
Receives a stream of prices in an input window called PriceFeed. The schema for this window has five columns: Id, Symbol, Price, Shares, and TradeTime. The window uses the Id field as a primary key, and is set to keep the last 10 minutes of price events:
<pre><code>CREATE INPUT WINDOW PriceFeed
SCHEMA
 (Id integer ,
   Symbol string ,
   TradeTime seconddate ,
   Price float ,
   Shares integer)
PRIMARY KEY (Id)
KEEP 10 MIN ;
</code></pre>
</td><td>


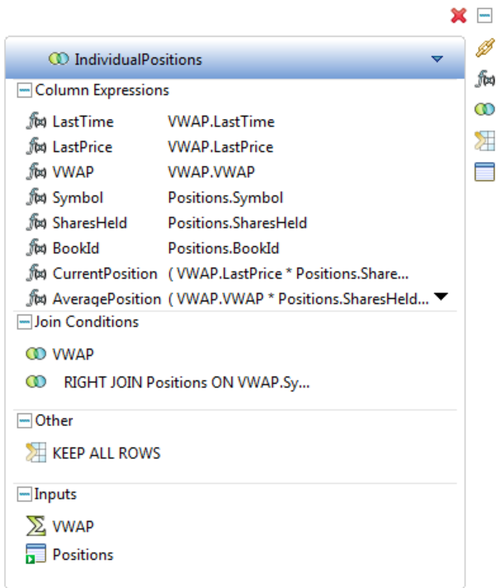
</td></tr>
<tr><td>
Applies an aggregate simple query to create a 10-minute moving average—a volume weighted average price (VWAP). With the VWAP, you can see the value of positions based on the average price, rather than see the value of your positions change with every small price movement. The VWAP formula is calculated as:
<pre><code>/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW VWAP
PRIMARY KEY DEDUCED
KEEP ALL ROWS
 AS
 SELECT
     PriceFeed.Symbol Symbol ,
     PriceFeed.TradeTime LastTime ,
     PriceFeed.Price LastPrice ,
     sum (PriceFeed.Price *
PriceFeed.Shares) /
      sum (PriceFeed.Shares) VWAP
 FROM PriceFeed
 GROUP BY PriceFeed.Symbol ;
</code></pre>
</td><td>


</td></tr>
</table>

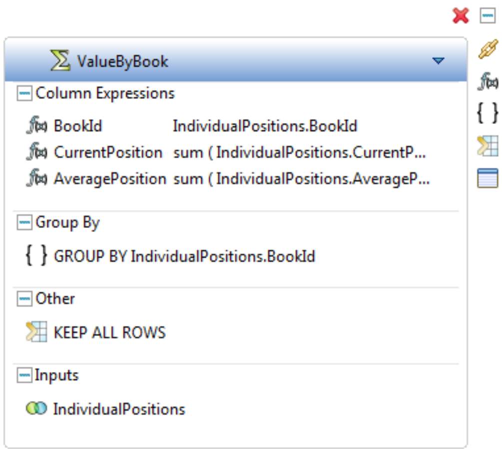
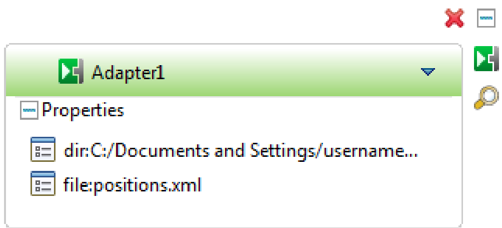
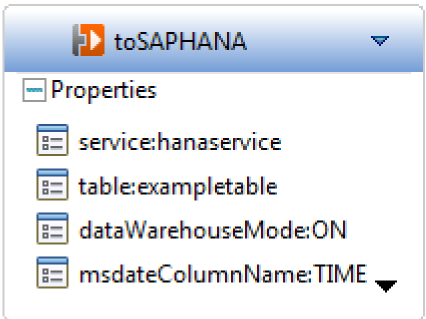
<table>
<tr><th>CCL Element</th><th>Diagram Element</th></tr>
<tr><td>
Reads data from another input window, Positions, with three columns: BookId, Symbol, and SharesHeld:

<pre><code>CREATE INPUT WINDOW Positions
  SCHEMA
   (BookId string ,
     Symbol string ,
     SharesHeld integer)
  PRIMARY KEY (BookId, Symbol) KEEP ALL
ROWS ;
</code></pre>
</td><td>


</td></tr>
<tr><td>
Applies a join simple query, joining the market price (from the VWAP aggregate) to your holdings (Positions), so that you can see the value of your position in each stock:

<pre><code>/**@SIMPLEQUERY=JOIN*/
CREATE OUTPUT WINDOW IndividualPositions
PRIMARY KEY DEDUCED
KEEP ALL ROWS
 AS
 SELECT
   VWAP.LastTime LastTime ,
   VWAP.LastPrice LastPrice ,
   VWAP.VWAP VWAP ,
   Positions.BookId BookId ,
   Positions.Symbol Symbol ,
   Positions.SharesHeld SharesHeld ,
   VWAP.LastPrice *
     Positions.SharesHeld CurrentPosition ,
   VWAP.VWAP *
     Positions.SharesHeld AveragePosition
 FROM VWAP RIGHT JOIN Positions ON
VWAP.Symbol = Positions.Symbol ;
</code></pre>
</td><td>


</td></tr>
</table>

<table>
<tr><th>CCL Element</th><th>Diagram Element</th></tr>
<tr><td>Applies one more aggregation to show the total value of each "book." This aggregate, ValueByBook, groups current and average values for individual positions into different "books." Each book may comprise a set of investment portfolios or funds. In the CCL, a GROUP BY clause performs the aggregation:
<pre>
/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW ValueByBook
PRIMARY KEY DEDUCED
KEEP ALL ROWS
 AS
  SELECT
     IndividualPositions.BookId BookId ,
     sum
(IndividualPositions.CurrentPosition)
           CurrentPosition ,
     sum
(IndividualPositions.AveragePosition)
           AveragePosition
  FROM IndividualPositions
  GROUP BY IndividualPositions.BookId ;
</pre></td><td>
</td></tr>
<tr><td><pre>
ATTACH INPUT ADAPTER Adapter1
TYPE toolkit_file_xmllist_input
TO
Positions
PROPERTIES dir =
'<SAP HANA-studio-workspace>\exampledata',
file = 'positions.xml' ;
</pre></td><td>
</td></tr>
<tr><td>Attaches an output adapter to output the processed data to an SAP HANA table. This adapter requires an existing connection to SAP HANA, which is created using the SAP HANA service from the Data Services view. See Adding a Connection to an SAP HANA Database in the SAP HANA Smart Data Streaming: Configuration and Administration Guide for detailed instructions on creating the service.
<pre>
ATTACH OUTPUT ADAPTER toSAPHANA TYPE
hana_out
TO ValueByBook
PROPERTIES service = 'hanaservice' ,
table = 'exampletable' ,
dataWarehouseMode = 'ON' ,
msdateColumnName = 'TIME' ;
</pre></td><td>
</td></tr>
</table>

Column Alias in Expressions

Each expression defines a unique name or alias for the column.

In the portfolio valuation sample project, a derived window called VWAP takes input from an input stream (PriceFeed) with columns Symbol, Price and TradeTime, and it includes an aggregate expression. Columns aliases for this derived window (created in visual editor as an aggregate simple query) are the following:

Alias	Column Expression
Symbol	PriceFeed.Symbol
LastPrice	PriceFeed.Price
VWAP	(sum ((PriceFeed.Price * CAST (FLOAT , Price-Feed.Shares))) / CAST (float , sum (Price-Feed.Shares)))
LastTime	PriceFeed.TradeTime

Parent topic: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Next task: Creating the Project [page 304]

Related Information

Sample Project with Modules [page 326]

12.2 Creating the Project

In studio, define a new set of processing instructions for event data.

Prerequisites

Start SAP HANA studio.

Procedure

1. Select **File** > **New** > **Project...**, then select **SAP HANA smart data streaming** > **New streaming Project**.
2. For the purposes of this tutorial, in the **Name** field, enter `my_portfolio_valuation`.
3. In the **Directory** field, accept the default location or browse to a directory in which to store the new project folder.
   Smart data streaming creates three files in the named directory:

`<project-name>.ccl`	contains the CCL code.

<project-name>.cclnotation	contains the diagram that corresponds to the .ccl file.
<project-name>.ccr	contains the project configuration.

For example, for a project directory named "trades," smart data streaming creates a trades.ccl, trades.cclnotation, and trades.ccr file in the trades directory.

4. Click **Finish** to create the project files.
The new project opens in the visual editor with one input stream, NEWSTREAM, and an inline schema ready for editing.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous: About the Portfolio Valuation Sample Project [page 300]

Next task: Editing a Project Diagram [page 305]

12.3 Editing a Project Diagram

Edit projects in the visual editor by adding shapes from the palette to the project diagram, connecting them, and completing the configuration of each shape.

Procedure

1. If the sample project diagram is not already open in the visual editor, open it now:
   a. In SAP HANA Streaming Development perspective, from project explorer, open the sample project, my_portfolio_valuation.
   b. Navigate to the .cclnotation file in your project folder and double-click my_portfolio_valuation.cclnotation.
2. Click in the diagram to begin editing using the palette.

   > **Tip**
   >
   > To make the visual editor window full-screen, double-click the <name>**:Diagram** tab at the top. Double-click again to revert.

3. Select the input stream element NEWSTREAM that was added automatically when you created the project, right-click, and choose **Delete Element**.
To run the sample project with example data, delete this element from the project before compiling.
   - **Delete Element** – removes the element from the project.
   - **Delete from Diagram** – removes the element from the diagram, but retains it in the project. When you run the project, everything in the project runs, even elements that are not on the diagram.
4. (Optional) To toggle between the visual editor and the CCL editor, choose **Switch to Text** or **Switch to Visual** (**F6**).

> **Note**
>
> The visual editor, like other graphical user interfaces, offers several ways to accomplish most tasks, although this guide may not list all of them. For example, in many contexts you can carry out an action by:
> - Clicking a button or other icon in a shape, or on the main toolbar
> - Using a shortcut key
> - Double-clicking an element to open it
> - Right-clicking to select from the context menu
> - Selecting from the main menu bar
> - Editing element values in the properties view

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Creating the Project [page 304]

Next task: Adding an Input Adapter [page 306]

12.4 Adding an Input Adapter

Attach an adapter by inserting it in the diagram, connecting it to a stream or window, and setting properties.

Context

This tutorial shows you how to insert an adapter, enable it for schema discovery, then generate and attach the input window and its schema automatically. This is the best practice for creating a schema when using an adapter that supports schema discovery.

Alternatively, SAP HANA smart data streaming allows you to create the stream or window and then attach an adapter. Use this method for adapters that do not support schema discovery, or to explicitly create an inline schema for input streams or windows.

Procedure

1. For this example, from within the SAP HANA Streaming Development perspective, open the **Input Adapters** compartment in the palette (to the right of the diagram) and select the **File/Hadoop Event XML Input** adapter, which reads data from an XML file.
2. Click the adapter in the palette, then click in the diagram.
   The adapter shape is inserted but its border is red, indicating it is not complete, until you define its properties and attach it to a stream or window.

3. In the adapter shape toolbar hovering outside the shape, click **Edit Properties** ( ) .
4. (Optional) In the Adapter Properties dialog, change **Name** to identify your adapter.
5. Configure the adapter for schema discovery:
   Required properties are in red.

   > **Note**
   >
   > Leave **Use named property set** unchecked, as this option does not allow you to discover the schema for this adapter.

   a. Click in the Value column for Directory and click the Browse button ( ).
   b. Click the Browse button in the Directory dialog to select the folder with the data files you want the adapter to read. Click OK.
   For this example, specify the absolute path to the sample data installed with the product.

Property	Value
Directory	`<SAP HANA-Studio-workspace/exampledata>` When you install the SAP HANA smart data streaming package, you install the `STREAMING_HOME/examples/ccl/exampledata` folder with it. If you have not done so already, copy the `exampledata` folder to your SAP HANA studio workspace.

   c. Enter a wildcard (*) in the Value column for File (in Directory). Click **OK**.
6. Click **OK**, then press **Ctrl+S** to save.

Next Steps

Import the schema and create a connected input stream or window with the same schema as the data file.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Editing a Project Diagram [page 305]

Next task: Discovering a Schema [page 308]

Related Information

Adapters [page 30]

12.5 Discovering a Schema

In studio, use the **Schema Discovery** button in the adapter shape to discover and automatically create a schema based on the format of the data from the adapter.

Prerequisites

Add the adapter to the diagram and set its properties.

Context

Rather than manually creating a new schema in your smart data streaming project, use schema discovery to discover and automatically create a schema, stream, or window based on the format of the data from the datasource to which your adapter connects. In this tutorial, you discover the schema for the PriceFeed input window from the File/Hadoop Event XML Input adapter, which corresponds to the schema of the source file.

Each adapter that supports schema discovery has unique properties that enable schema discovery. For a list of adapters that support schema discovery and which properties to configure, see Adapter Support for Schema Discovery [page 82].

Procedure

1. From within the SAP HANA Streaming Development perspective, click **Schema Discovery** on the adapter toolbar.
   Studio displays a Progress Information box and looks for the configuration.
   - If the schema is configured properly and one or more data sets are found, a Schema Discovery: Select Schema dialog appears where you can view and select a schema.
   - If the schema is not successfully discovered, an error message appears stating that no schema was discovered for the adapter. You can:
     - Check that the adapter properties are configured for schema discovery.
     - Check Adapter Support for Schema Discovery [page 82] to see if the adapter supports schema discovery.
2. Select the schema you need.

   You can expand the data set to view the schema.

   For this example, select **positions.xml**, then click **Next**.
3. In the Schema Discovery: Create Element dialog, choose **Create new input window (with inline schema)**.

   This option creates and attaches a new window to the adapter, creates an inline schema for the window, and populates the window with the schema discovered from the adapter. When the adapter is not yet attached to a stream or window, other options are the following:

Option	Description
Create a new input stream (with inline schema)	Creates and attaches a new stream to the adapter, creates an inline schema for the stream, and populates the stream with the schema discovered from the adapter.
Create a new input stream (with attached schema)	Creates and attaches a new stream to the adapter, creates and attaches a new named schema to the stream, and populates the stream with the schema discovered from the adapter.
Create a new input window (with attached schema)	Creates and attaches a new window to the adapter, creates and attaches a new named schema to the window, and populates the window with the schema discovered from the adapter.
Create new named schema	Creates a new named schema and populates it with the schema discovered from the adapter.

4. Click **Finish**.
    - The new input window appears with the default name positions_xml_window1, and is automatically connected to the File/Hadoop Event XML Input adapter.
    - The adapter file property is set. The red warning border disappears, indicating that the element is now valid.
5. In the schema compartment of the input window, click the **Toggle Key** buttons for the BookId and Symbol columns to specify the primary key.

    The key button indicates primary key columns. With the primary key, the shape becomes valid.
6. Set a keep policy in the input window:
    a. Click **Set Keep Policy**.
    b. In the Edit Keep Policy dialog, choose **All Rows**, and click **OK**.
7. Click the input window **Edit** button and name it `positions`, using lower case letters.

Next Steps

Create another input window, PriceFeed. Either:

- Create the PriceFeed input window manually, following steps in the next task, or,
- Insert another File/Hadoop Event XML Input adapter and configure it for schema discovery.
  This time, when you discover the schema, choose `pricefeed.xml` in the `exampledata` directory. Name the input window PriceFeed, click the Id column to make it the primary key, and set the keep policy to `10 MIN`.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Adding an Input Adapter [page 306]

Next task: Adding an Input Window Manually [page 310]

12.6 Adding an Input Window Manually

In studio, add an input window to the diagram in the sample project `my_portfolio_valuation`.

Context

These steps let you create an input window directly, and define the schema, without importing a schema.

If you used the input adapter to discover the schema and generated both input windows automatically, skip these steps and go directly to the next task.

Procedure

1. From within the SAP HANA Streaming Development perspective, in the visual editor, in the palette to the right of the diagram, open the **Streams and Windows** compartment.
2. Click **Input Window**.
3. Click in an empty area in the diagram where you want to insert the input window.
   The input window object is added to the project. The red border indicates that it needs more definition to be valid.
4. To set the name of the input window, either:
   - In iconic mode, click once to select the shape, then click again to edit the name.
   - In verbose mode, click the edit icon next to the name.

   For this example, enter the name `PriceFeed`.
5. Click the "plus" sign to expand the shape to verbose mode, if necessary, and click **Add Column** () on the toolbar in the input window, to add each new column.

   > Tip
   >
   > Hover over any icon to see its name.

   A new column is created with a default name, and default datatype of integer.
6. Specify additional columns.
   a. Double-click each column name to edit it.
   b. Double-click each datatype to select the correct datatype.

   For this example, enter these column names and datatypes:

Id	integer
Symbol	string
TradeTime	seconddate

Price	float
Shares	integer

7. Click the button for the Id column to toggle it to the Key symbol.

   Input windows require a primary key.

   The Id column is now the primary key for the PriceFeed input window. The red warning border disappears, indicating that the element is now valid.
8. Create a retention window.
   a. Click **Set Keep Policy** .
   b. In the Edit Keep Policy dialog, select **Time**, and enter `10 MIN` in the text box. Click **OK**.

   The default policy is to keep all rows of incoming data.

   This step defines a KEEP clause, and retains all price events received in the last 10 minutes. Without a KEEP clause, the PriceFeed window would grow infinitely large. See Retention Policies [page 25].
9. Save (**Ctrl+S**).

   This saves changes to both the `.cclnotation` file (the diagram) and the `.ccl` file (the CCL).

Results

The input window and its schema (or deduced schema) are in the diagram.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Discovering a Schema [page 308]

Next task: Creating an Aggregate as a Simple Query [page 311]

12.7 Creating an Aggregate as a Simple Query

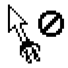
In studio, add an aggregate simple query to the sample diagram to create a volume weighted average price (VWAP).

Context

An aggregate query groups events that have common values, and computes summary statistics for the group.

Procedure

1. From within the SAP HANA Streaming Development perspective, in the visual editor palette, in **Streams and Windows**, click **Aggregate**.
2. Click in the diagram to create the object.
3. Change the default name, Aggregate1, to VWAP.
4. Connect PriceFeed to the VWAP aggregate:
   a. Click the **Connector** tool in the palette.
   b. Click the **PriceFeed** input window, then click the **VWAP** aggregate.
   Click the shape that produces the output first, then the shape that receives the data, to indicate the direction of data flow. Watch for visual indicators that show you when the connection is valid:

Indicator	Meaning
	Connection is allowed
	Connection is not allowed

5. Enter column expressions:
   a. Click **Add Column Expressions** (), then **Copy Columns from Input** () in the shape toolbar to select the columns to copy into the schema for the aggregate window.
   For this example, copy these columns:
      - PriceFeed.Symbol
      - PriceFeed.TradeTime
      - PriceFeed.Price
   b. Edit column names to clarify that these columns will hold the most recent price and time for the group:
      - Change TradeTime to LastTime
      - Change Price to LastPrice
   c. Add additional columns by clicking **Add Column Expression** in the shape toolbar.
   For this example, add another column and edit its name to `VWAP`.
6. Edit a column expression by double-clicking to open the inline editor, by selecting one of the tabs in the Properties view, or by selecting an expression and pressing **Ctrl+F2** to edit it using the pop-up expression editor.
   For this example, edit the VWAP column expression to:

```
sum ( PriceFeed.Price *
PriceFeed.Shares ) /
sum ( PriceFeed.Shares )
```

7. Click **Add Group Clause** (**{ }**) to edit the grouping of columns in the aggregate object.

> **Note**
>
> The aggregate shape must have exactly one GROUP BY expression.

For this example, select **PriceFeed.Symbol** as the grouping column.

The red warning border disappears, indicating that the element is now valid.

8. Set a keep policy in the input window:
    a. Click **Set Keep Policy** .
    b. In the Edit Keep Policy dialog, choose **All Rows**, and click **OK**.

Results

By default, the aggregate is created as output, which allows external applications to subscribe to or query it, and allows you to view it using the Stream View in the SAP HANA Streaming Run-Test perspective.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Adding an Input Window Manually [page 310]

Next task: Creating a Join as a Simple Query [page 313]

12.8 Creating a Join as a Simple Query

Add a join to the sample project. A join combines events from two or more inputs to create a single stream or window. It is similar to a join in SQL.

Context

Smart data streaming supports inner joins, left and right outer joins, and full outer joins, with join syntax comparable to SQL ANSI join syntax and comma-separated syntax. For more information, see Joins [page 183].

Procedure

1. From within the SAP HANA Streaming Development perspective, in the visual editor palette, in **Streams and Windows**, select **Join**.

    If necessary, close the compartments below **Streams and Windows**, or use the arrow below the compartment, so that **Join** is visible.
2. Click in the diagram to create the object.
    For this example, name the the join object `IndividualPositions`.

3. Using the connector tool, connect the join object to the appropriate stream or window.
   Attach join objects to any stream, window, or Flex operator. Join objects have multiple inputs, but only one output.

   > **Note**
   >
   > Streams, windows, or keyed streams can participate in a join. Only one stream can participate in a join at a time.

   For this example, connect the VWAP aggregate object and the Positions input window to the IndividualPositions join object, in that order.

   > **Tip**
   >
   > To add multiple connections, **Shift+click** while holding the **Connector** tool. To return to normal selection, press **Esc** or click the **Select** tool in the palette to release it.

4. Click **Add Column Expressions** (), then **Copy Columns** () in the join shape toolbar and select columns to copy.

   > **Tip**
   >
   > If you get an error, or do not see all columns from both inputs listed, reconnect the new join element to the Positions or VWAP shapes as needed.

   For this example, choose **Select All**, then clear the check box on **VWAP.Symbol** so that you do not get the symbol field twice.
5. Click **Add Column Expressions** ().
   For this example add two columns: CurrentPosition and AveragePosition.
6. To modify column expressions, do one of the following:
   - Double-click on the expression to open the inline editor, and either type directly or press **Ctrl+Space** for syntax completion assistance, to pick from column names and functions, or,
   - Press **Ctrl+F2** to open the expression editor. Press **Ctrl+Space** to display the available input columns and built-in functions, or enter the desired expression manually, or,
   - Modify the expression in the Properties view.

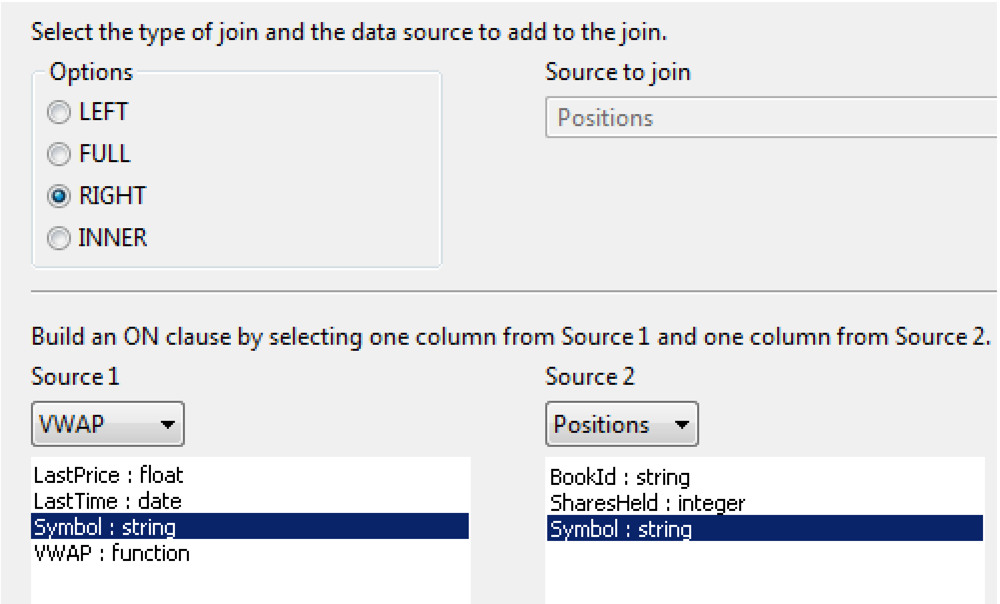
   For this example, create these Column Expressions:
   - CurrentPosition: `( VWAP.LastPrice * positions.SharesHeld )`
   - AveragePosition: `( VWAP.VWAP * positions.SharesHeld )`
7. In the Join Conditions compartment of the join shape, set up the join conditions.
   If you connected the join to the VWAP and Positions inputs, in that order, there are now two elements in the Join Conditions compartment. The first defines the leftmost element for the join. If you connected to VWAP first, the first element (left side of the join) is VWAP. For this example, configure the second join element.
   a. Double-click the second join element to open the Edit Join Expression dialog.
   b. Choose a join type.
      For this example, use **RIGHT**, which is a right outer join. You want RIGHT because VWAP is the first, or left input, and Positions is the second, or right input. You only want your positions in the output; you do not need prices for symbols that are not held in the portfolio.

c. Select the columns to join on.

You cannot edit join constraints manually in the visual editor.

For this example:.

- As Source 1, ensure that **VWAP** is in the dropdown, and select the option starting with **Symbol:** as the column.
- As Source 2, ensure that **Positions** is in the dropdown, and select the option starting with **Symbol:** as the column.

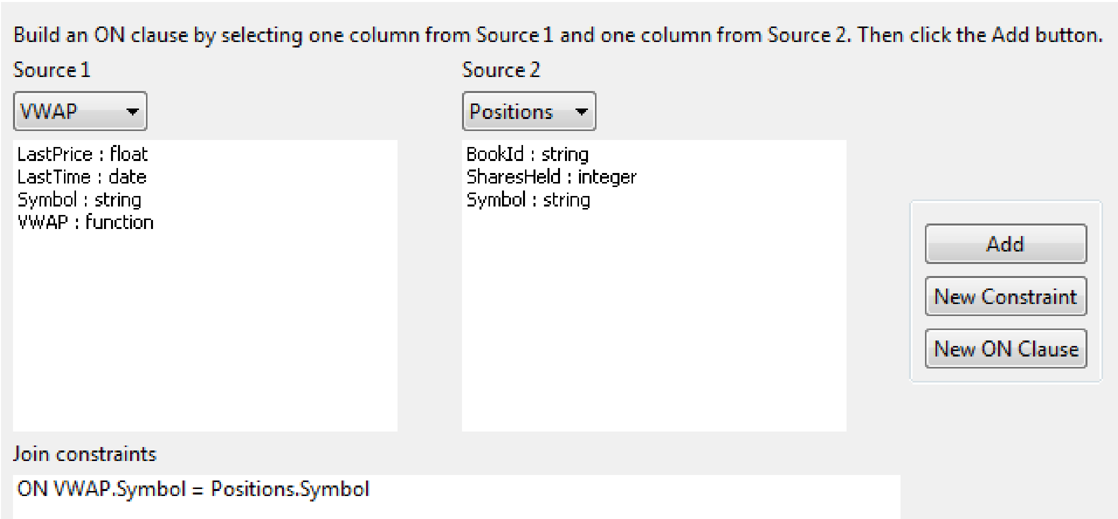


d. Click **Add**. If a message displays indicating that datatypes for the selected columns should match, click **Yes** to continue.

The columns chosen appear in Join Constraints, where you now see:

```
ON VWAP.Symbol=positions.Symbol
```

The dialog shows:

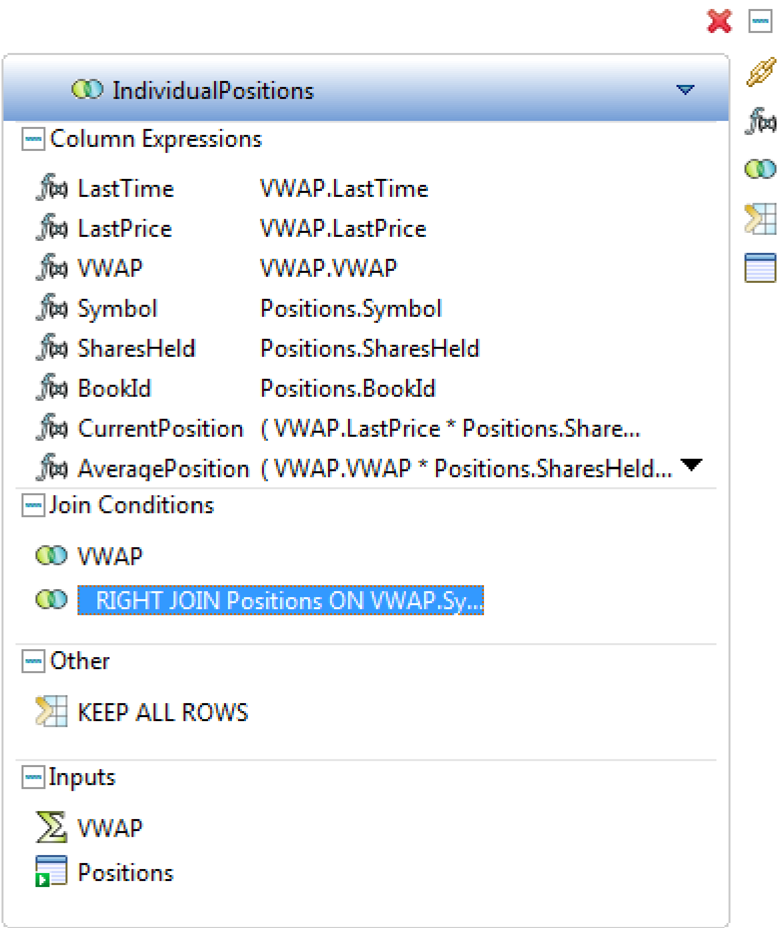


e. Click **OK**.

8. Set a keep policy in the input window:
    a. Click **Set Keep Policy** .
    b. In the Edit Keep Policy dialog, choose **All Rows**, and click **OK**.
9. In the join shape, click (Toggle Type to OUTPUT). If this option is not available, the shape is already set to OUTPUT.

Results

The IndividualPositions join shape now shows the completed join, as shown in the figure.



Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Creating an Aggregate as a Simple Query [page 311]

Next task: Preparing Data for Output [page 317]

12.9 Preparing Data for Output

Add a new aggregate to finalize the data and prepare it for output into SAP HANA.

Context

Create an additional Aggregate Simple Query and name it ValueByBook.

Procedure

1. Connect the IndividualPositions join object to it.
2. Click **Copy Columns** () in the shape toolbar and copy columns BookId, CurrentPosition, and AveragePosition.
3. Set column expressions:
   - `BookId IndividualPositions.BookId`
   - `CurrentPosition sum ( IndividualPositions.CurrentPosition )`
   - `AveragePosition sum ( IndividualPositions.AveragePosition )`

   > Tip
   >
   > Use the inline editor. Double-click on the column expression, and use the **Home** and **End** keys to quickly edit the expression.
4. Add the GROUP BY clause (**{ }**) `IndividualPositions.BookId`.
5. Toggle to OUTPUT if the shape is not set as OUTPUT already.

Next Steps

Set up a connection and add an output adapter to deposit data into an SAP HANA studio table.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Creating a Join as a Simple Query [page 313]

Next task: Adding an Output Adapter for SAP HANA [page 318]

12.10 Adding an Output Adapter for SAP HANA

Create an SAP HANA service definition, and insert and configure an SAP HANA Output adapter using the studio.

Prerequisites

- You have installed the SAP HANA client driver on the same machine as studio and set up an SAP HANA database.
- You have created a connection to the smart data streaming server through the SAP HANA Streaming Run-Test perspective server view.

> **Note**
>
> The smart data streaming server that you connect to must be version 5.1 SPS 10 or newer. If you connect to an older version, the option to create a HANA service does not appear in the Data Services view.

Procedure

1. In the data services view, right-click on the streaming server and select **Discover**.
2. Right-click on the workspace you want to use and select **Add HANA Service**.
3. Right-click the new database service and select **Rename Data Service**. For this tutorial, name it `hanaservice`.
4. In the properties view, set the **User** parameter to the username necessary for communicating with the database.
5. Set the **Password** parameter to the password for your username.
6. (Optional) Add a description of the database service in the **Description** parameter.
7. (Optional) Select **Use Default HANA Server** if you are connecting to the local SAP HANA server.

   If you choose to connect to the local SAP HANA server, you need not provide an SAP HANA hostname or instance number.
8. (Required if connecting to a remote SAP HANA database) Specify a database type:
   - Select **Single Tenant** if the SAP HANA system is the only database at the location, or;
   - Select **Multi Tenant** if you are connecting to a system with multiple databases. If you select this option, specify the name of the database that you are connecting to in the **Database Name** field.
9. (Required if connecting to a remote SAP HANA database) Set the **HANA Hostname** parameter to the fully qualified domain name (FQDN) of the host on which the SAP HANA database is installed.
10. (Required if connecting to a remote SAP HANA database) Set the **HANA Instance Number** parameter to the instance number of the target database.
11. (Optional) Enable **Multi-Byte Character Support** if the database you are connecting to uses multi-byte characters.
12. In the data services view, right-click on the streaming server and select **Discover**.
13. Right-click on the workspace you want to use and select **Add ODBC Service**.
14. Click anywhere outside the Properties view to save your work.
15. Open the **Output Adapters** compartment in the palette and choose **HANA Output**. This adapter uses multiple parallel ODBC connections to load information from smart data streaming into the SAP HANA server.
    a. Add the adapter to the diagram and name it `toSAPHANA`.
    b. Click **Edit Properties** ( ). Fill in those property values in red:
       - Set **Database Service Name** to `hanaservice`.
       - Set **Target Database Table Name** to the required SAP HANA table. In this example, you will not be configuring the project from the SAP HANA side, so enter `exampletable` in the value field.
    c. Click **OK**, then press **Ctrl+S** to save.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Preparing Data for Output [page 317]

Next task: Preparing to Compile [page 320]

Related Information

Adapters [page 30]

12.11 Preparing to Compile

Clean up the diagram by removing unused elements.

Procedure

1. Delete any unused elements from the project so that you can run it.
   For example, if you have not done so, remove the unused input stream element NEWSTREAM that was added automatically when you created the project.
2. (Optional) Toggle to Iconic mode or Verbose mode:
   - Click the Toggle Image button in the upper left corner of a shape, or,
   - Click the All Iconic or All Verbose button in the toolbar.
3. (Optional) Click Layout Left to Right to line up shapes.
4. (Optional) To close the diagram, press **Ctrl+W** or **Ctrl+F4**, or click the **X** on the tab at the top of the editor .

Results

The completed diagram should look like this figure in Verbose mode. You may need to open some compartments and click again to see details for all elements.

> **Note**
>
> Even though it does not appear below, if you added the SAP HANA Output adapter to the project, the adapter is also visible in the diagram.

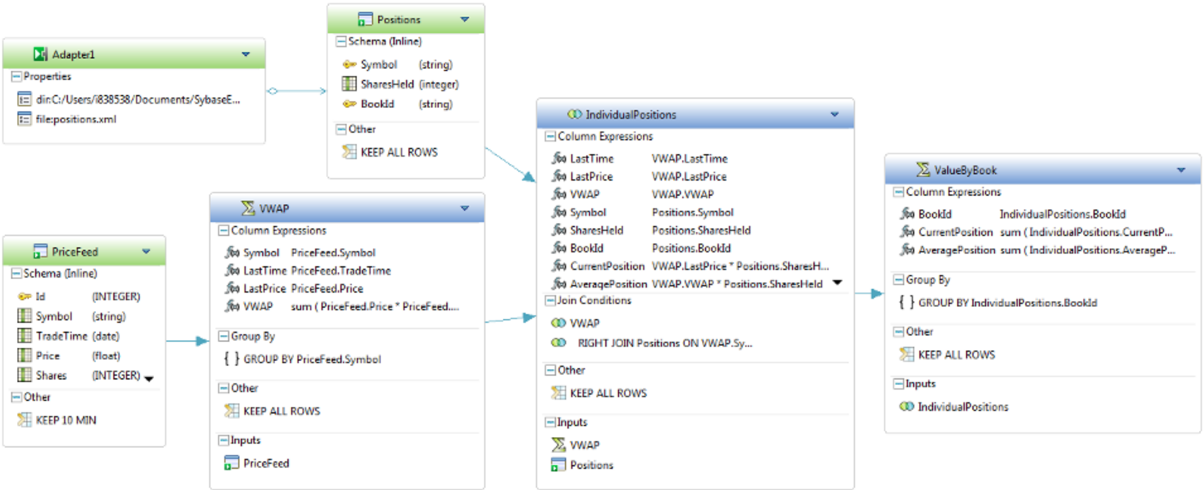


Figure 8: Completed Sample Portfolio Valuation Diagram

Next Steps

Compile the project and check for errors.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Adding an Output Adapter for SAP HANA [page 318]

Next task: Compiling the Project and Viewing Problems [page 321]

12.12 Compiling the Project and Viewing Problems

Compile a project in studio before running it to check for errors and make corrections. Use the Problems view to view error details.

Procedure

1. Update the Default Server URL preference:
   a. Select **Window** > **Preferences**.
   b. In the Preferences dialog, select **SAP HANA smart data streaming**.
   c. In the **Default Server URL** field, click **Change** and select a server from the Select Default Server URL dialog. Click **OK**.
   d. In the Preferences dialog, click **Apply**, then **OK**.
2. To compile the project, either:

- Click the **Compile Project** button in the main toolbar, or,
- Press **F7**.

The project compiles and reports any errors found. Compilation errors are displayed in the Problems or Console view, depending on the type of error.

3. Click on a problem in the Problems view, or expand the group to see individual errors.

   By default, the Problems view is at the bottom of the screen, and problems are grouped by severity.

   Error details appear in the Problems view and in the status bar at the bottom left side of the screen.

   > **Tip**
   >
   > If you double-click on a problem in the Problems view while the project is open in the visual editor, the CCL editor opens read-only to show you where the problem is. To fix the problem, either:
   > - Return to the visual editor and fix it there, or,
   > - Close both the visual editor and CCL editor for the project, and then reopen the project in the CCL editor.

4. If the error message is too long to show the entire message, click it to read the full text in the status bar at the bottom of the studio window.
5. Right-click an item to choose from the context menu:

Option	Action
Go to	Highlight the problem in the .ccl file. The CCL editor opens in read-only mode.
Copy	Copy error details to the clipboard. When you exit studio, the contents of problems view are removed. Use this option to save off errors.
Show in	Display details in Properties view.
Quick Fix	(Disabled)
Properties	Display details in a dialog box.

6. (Optional) Click the View dropdown menu to see more options.
7. Click the **Console** tab to view compiler results.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Preparing to Compile [page 320]

Next task: Deploying the Sample Project [page 323]

12.13 Deploying the Sample Project

Run the project and watch it open in the SAP HANA Streaming Run-Test perspective.

Prerequisites

Make sure the project compiles without errors. Correct any problems before you run the project.

Procedure

1. With the diagram open in the visual editor in the SAP HANA Streaming Development perspective, click in the main toolbar.
2. Review the running project in SAP HANA Streaming Run-Test perspective.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Compiling the Project and Viewing Problems [page 321]

Next task: Loading Data into the Sample Project [page 323]

12.14 Loading Data into the Sample Project

Test the sample project by loading reference data into the Positions window.

Context

If your project has the File/Hadoop Event XML Input adapter attached to the Positions input window, data is loaded automatically when you start the project. If you removed or omitted the adapter, use this alternative process to load the sample data.

Procedure

1. From within the SAP HANA Streaming Run-Test perspective, in the server view, expand the **my_portfolio_valuation** project to show the list of windows and streams in the project.

2. Double-click the Positions input window to open it in stream view.
   Stream view is in the upper right-hand portion of the SAP HANA Streaming Run-Test perspective.
   - If your diagram has the File/Hadoop Event XML Input adapter connected to the Positions input window, stream view shows sample data for Positions, loaded automatically from the adapter.
   - If you removed the adapter, go to the next step to load the data manually.
3. Load positions data from a file into the Positions window.
   a. Go to file upload view.
   b. Click the Select Project button in the view toolbar. If you only have one project open in server view, no dialog appears and smart data streaming automatically selects the project for you. Otherwise, select the **my_portfolio_valuation** project in the dialog, and click **OK**.
   c. Click the **Browse** button, navigate to your <SAP HANA-studio-workspace>\exampledata folder, and select positions.xml.
      When you install the SAP HANA smart data streaming package, you install the SAP_HANA_STREAMING/examples/exampledata folder with it. If you have not done so already, copy the exampledata folder to your SAP HANA studio workspace.
   d. Click **Open**.
   e. With positions.xml highlighted in File Upload view, click the **Upload** button.

   Watch the data flow in stream view, as studio loads the three positions for Book1 and Book2.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Deploying the Sample Project [page 323]

Next task: Testing the Project with Recorded Data [page 324]

12.15 Testing the Project with Recorded Data

Play back the previously recorded price feed data, and view the continuous portfolio valuations in the sample project.

Procedure

1. From within the SAP HANA Streaming Run-Test perspective, in the server view, double-click the IndividualPositions, VWAP, and ValueByBook output windows.

   In the server view list, a red arrow in the lower right corner of the window icon ( ) indicates the output windows.
2. Click the **Playback** tab.
3. If necessary, click the **Select Project** button in the upper right corner of playback view.
   - If you only have one project running, studio selects it for you.

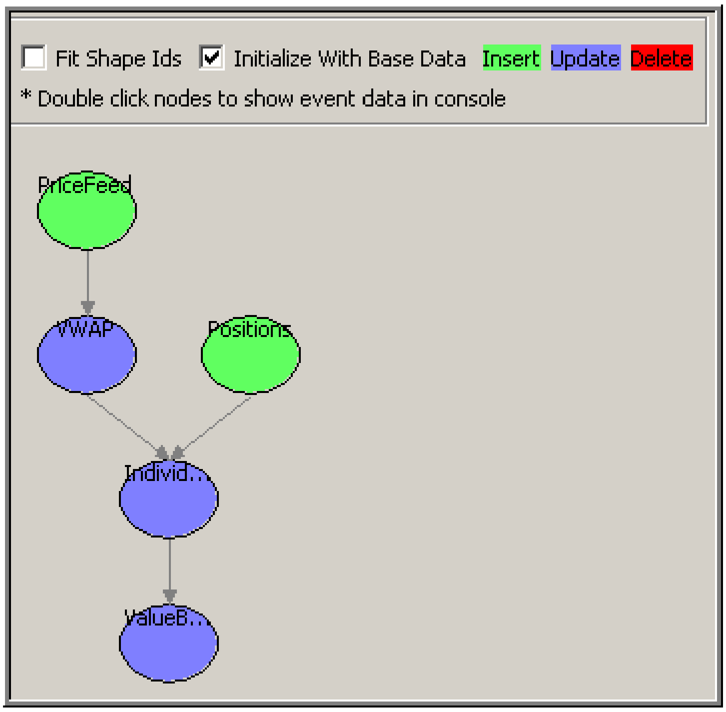
- Otherwise, select the **my_portfolio_valuation** project in the dialog and click **OK**.

4. Click the **Select Playback File** button.
5. Navigate to your `<SAP HANA-Studio-workspace>\exampledata` folder, and select `pricefeed.xml`. Click **Open**.

   When you install the SAP HANA smart data streaming package, you install the `SAP_HANA_STREAMING/examples/exampledata` folder with it. If you have not done so already, copy the `exampledata` folder to your SAP HANA studio workspace.
6. In the playback view, in the playback mode frame, click the **rec/ms** button, then enter a **rec/ms** value of 1.

   A value of 1 plays back at a rate of 1000 records per second.
7. Click the green **Start Playback** button to start playback of the price feed.
8. While the data plays back, click each of the output windows in stream view to see the calculations revised in real-time.
9. (Optional) In the event tracer view, choose **Select Running Project**, and click **Initialize with Base Data**. In this example, Event Tracer shows the PriceFeed and Positions elements in green to indicate Insert operations. VWAP, IndividualPositions, and ValueByBook are in blue, indicating Updates. Colors change as different event types are processed.

   Double-click each node to watch event data in the Console.



10. To stop the playback, click **Stop** .
11. When you are done testing the project, right-click it in server view and choose **Stop Project**.

    If you omit this step, the project stops when you exit studio, but you may get an error.

> **Tip**
>
> If you see an error when you restart studio, or when you try to open a `.ccl` file after running a project, there may be multiple instances of studio trying to use the same studio workspace location. If this occurs, close studio and restart it.

Task overview: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Loading Data into the Sample Project [page 323]

Next: Sample Project with Modules [page 326]

Related Information

Project Execution and Testing [page 56]

12.16 Sample Project with Modules

This variation of the portfolio valuation project provides a complex, real-world example. The project uses a defined module with a named schema to easily scale out the application in a very high volume deployment. The SAP HANA Output adapter is not included in this module.

The module, `valuation.ccl`, computes the VWAP aggregate, and creates a join to the Positions window. The project uses the module to divide the moving data into smaller partitions, based on the first letter of the Symbol column. This strategy spreads the load out to more cores, thereby increasing throughput. By using modules, with very little coding you can easily double, quadruple, and so on, the number of partitions.

This example also implements the streaming tick data in PriceFeed as a stream rather than as an input window. Because keeping every tick would use a lot of memory, and because the state is never updated or queried, a stream is a more likely choice than a window in a real-world scenario for this event stream.

> **Example**
>
> Create Module valuation
>
> The valuation module:
>
> 1. Defines the input stream TradesIn.
> 2. Defines a stream, Filter1, that filters TradesIn data into a substream based on the declared parameters `afrom` and `ato`.
> 3. Defines the input window Portfolio.
> 4. Defines the VWAP aggregate as an output window.

5. Defines another output window, ValueBySymbol, that performs a join similar to the join simple query in the simple PortfolioValuation project, with the addition of a `cast` for the `float` data.

```
CREATE MODULE valuation
IN TradesIn,Portfolio
OUT ValueBySymbol,   VWAP
BEGIN
    IMPORT 'import.ccl';

    DECLARE
        PARAMETER STRING afrom;
        PARAMETER STRING ato;
    END;

    CREATE INPUT STREAM TradesIn
       SCHEMA TradesSchema ;
    CREATE  STREAM Filter1  AS
       SELECT * FROM TradesIn
         WHERE substr(TradesIn.Symbol,1,1) >= afrom
               and substr(TradesIn.Symbol,1,1) <= ato
    ;
    CREATE INPUT WINDOW Portfolio
       SCHEMA PortfolioSchema
       PRIMARY KEY (BookId, Symbol);
    CREATE OUTPUT WINDOW VWAP
       PRIMARY KEY DEDUCED AS
       SELECT Filter1.Symbol Symbol ,
       (sum((Filter1.Price * cast(FLOAT ,Filter1.Shares))) /
          cast(FLOAT ,sum(Filter1.Shares)))
       AS VWAP,
       sum (Filter1.Shares ) Total_Shares ,
       valueinserted(Filter1.Price) LastPrice,
       valueinserted(Filter1.TradeTime) TradeTime
       FROM Filter1
       GROUP BY Filter1.Symbol ;
    CREATE OUTPUT WINDOW ValueBySymbol
        SCHEMA (BookId STRING, Symbol STRING, CurrentPosition FLOAT,
AveragePosition FLOAT)
        PRIMARY KEY (BookId, Symbol) AS
        SELECT
            Portfolio.BookId  AS BookId,
            Portfolio.Symbol  AS Symbol,
            (VWAP.LastPrice * cast(FLOAT ,Portfolio.SharesHeld))
                AS CurrentPosition,
            (VWAP.VWAP * cast(FLOAT ,Portfolio.SharesHeld))
                AS AveragePosition
            FROM Portfolio JOIN
              VWAP
                 ON Portfolio.Symbol = VWAP.Symbol;
END;
```

Example

Create Named Schema TradesSchema

```
CREATE SCHEMA  TradesSchema
    ( Id integer ,
    Symbol string ,
    TradeTime seconddate ,
    Price float ,
    Shares integer )  ;
```

Example

Create Named Schema PortfolioSchema

```
CREATE SCHEMA PortfolioSchema
    ( BookId string ,
    Symbol string ,
    SharesHeld integer ) ;
```

Example

Import and Load the valuation Module

In the parent scope, the valuation module is loaded three times, as Valuation1, Valuation2, and Valuation3.

1. The IN clause binds the input streams in the module to streams in the parent scope. TradesIn is bound to InputStream1, and Portfolio is bound to InputPositions.
2. The OUT clause binds the output window in the module, ValueBySymbol, with the three parameterized output windows, VbySym1, VbySym2, and VbySym3, and partitions the VWAP aggregate as VWAP1, VWAP2, and VWAP3.

InputStream1	Input stream based on the imported schema, TradesSchema.
InputPositions	Input window based on the imported schema, PortfolioSchema.
UnionVWAP	Output window created as a UNION of the partitioned VWAP aggregate.

```
IMPORT 'import.ccl';
IMPORT 'valuation.ccl';
DECLARE
    PARAMETER STRING afrom :='A';
    PARAMETER STRING ato := 'Z';
END;
CREATE INPUT STREAM InputStream1 SCHEMA TradesSchema  ;
CREATE INPUT WINDOW InputPositions
  SCHEMA PortfolioSchema PRIMARY KEY ( BookId , Symbol ) ;
  LOAD MODULE valuation as Valuation1
    in TradesIn = InputStream1, Portfolio = InputPositions
    OUT ValueBySymbol = VbySym1,  VWAP = VWAP1
    PARAMETERS afrom = 'A', ato = 'J'
    ;
  LOAD MODULE valuation as Valuation2
    in TradesIn = InputStream1, Portfolio = InputPositions
    OUT ValueBySymbol = VbySym2, VWAP = VWAP2
    PARAMETERS afrom = 'K', ato = 'Q'
    ;
  LOAD MODULE valuation as Valuation3
    in TradesIn = InputStream1, Portfolio = InputPositions
    OUT ValueBySymbol = VbySym3,  VWAP = VWAP3
    PARAMETERS afrom = 'R', ato = 'Z'
    ;
CREATE OUTPUT WINDOW UnionVWAP
 PRIMARY KEY DEDUCED
 AS SELECT * FROM VWAP1
    UNION SELECT * FROM VWAP3
    UNION SELECT * FROM VWAP2 ;
CREATE OUTPUT WINDOW ValueBySymbol
 PRIMARY KEY (BookId,Symbol)
 AS SELECT * FROM VbySym1
    UNION SELECT * FROM VbySym3
    UNION SELECT * FROM VbySym2 ;
// ----------------------------
//  stream ValueByBook
```

```
CREATE  OUTPUT  WINDOW ValueByBook
    SCHEMA (BookId STRING, CurrentPosition FLOAT, AveragePosition FLOAT)
     PRIMARY KEY DEDUCED AS
    SELECT ValueBySymbol.BookId AS BookId,
        sum(ValueBySymbol.CurrentPosition) AS CurrentPosition,
        sum(ValueBySymbol.AveragePosition) AS AveragePosition
    FROM ValueBySymbol
    GROUP BY ValueBySymbol.BookId;


ATTACH INPUT ADAPTER Adapter1 TYPE toolkit_file_xmllist_input TO InputStream1
GROUP nostartGroup
PROPERTIES
dir = '../exampledata' ,
file = 'pricefeed.xml' ,
pollingPeriod = 0 ,
xmllistSecondDateFormat = 'yyyy-MM-ddTHH:mm:ss' ,
xmllistMsDateFormat = 'yyyy-MM-ddTHH:mm:ss';
ATTACH INPUT ADAPTER Adapter2 TYPE toolkit_file_xmllist_input TO InputPositions
PROPERTIES
dir = '../exampledata' ,
file = 'positions.xml' ,
pollingPeriod = 0 ,
xmllistSecondDateFormat = 'yyyy-MM-ddTHH:mm:ss' ,
xmllistMsDateFormat = 'yyyy-MM-ddTHH:mm:ss';
ADAPTER START GROUPS nostartGroup nostart   ;
```

Note

Edit the default value for the `dir` property to the sandbox base directory. For example, change `dir='../exampledata` to `dir='<sandbox-base-directory>/exampledata'`. Otherwise, the example does not compile properly and you receive an error message.

Parent topic: Appendix: Tutorial for Building and Testing a Project in Studio [page 299]

Previous task: Testing the Project with Recorded Data [page 324]

Related Information

About the Portfolio Valuation Sample Project [page 300]
Modularity [page 146]

13 Appendix: Performance and Tuning Tips

Optimizing performance in SAP HANA smart data streaming requires tuning at the project level as well as at the infrastructure level (machine, OS, network configuration, and so on).

If you tune your projects to produce maximum throughput and minimum latency but do not configure your infrastructure to handle the throughput, you will see sub-optimal performance. Likewise, if you configure your infrastructure to handle maximum throughput but do not tune your projects, your performance suffers.

In this section:

13.1 Distributing Load through Parallelization

To improve performance of large smart data streaming projects, separate the data into smaller chunks that are processed within their own partitions. Processing on multiple partitions in parallel can improve performance over processing in one large partition.

There are various ways to parallelize your smart data streaming project.

1. Application-based Partitioning

You can send all incoming data to each of the input adapters within your smart data streaming project, and then attach each of these adapters to a stream that filters a subset of the overall incoming data. The output adapters receive this data and output it to the external datasource.

Advantages:

- Improves performance and processes high volumes of data since having multiple streams processing subsets of the data divides the load on the processor.
- No need to create a custom adapter or do any custom coding aside from specifying the filtering.
- Allows for partitioning across cores, but this type of partioning is best suited across machines.

Disadvantages:

- Need to duplicate the input data feeding into the input adapters.

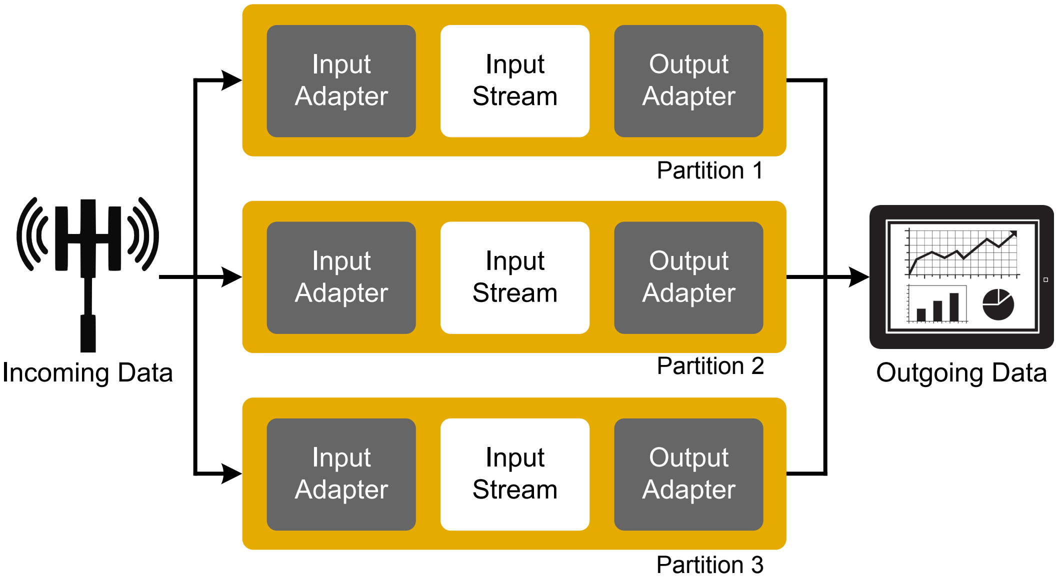


Figure 9: Application-based Partitioning

2. Partitioning Using a Custom Adapter

You can write a custom adapter to receive input data and publish it to various streams, keyed streams, or windows on separate machines. These streams or windows process and send this data to separate output adapters, which then publish it to the end datasource. The custom adapter is responsible for partitioning the input data in this scenario.

Advantages:

- Improves performance and processes high volumes of data by filtering incoming data across multiple machines.
- Adapters are customizeable to meet partitioning requirements.
- No need to duplicate any data.
- Allows for partitioning across cores, but this type of partioning is best suited across machines.

Disadvantages:

- Requires more effort in terms of coding because you create a custom adapter. You cannot currently partition the available adapters provided with smart data streaming.

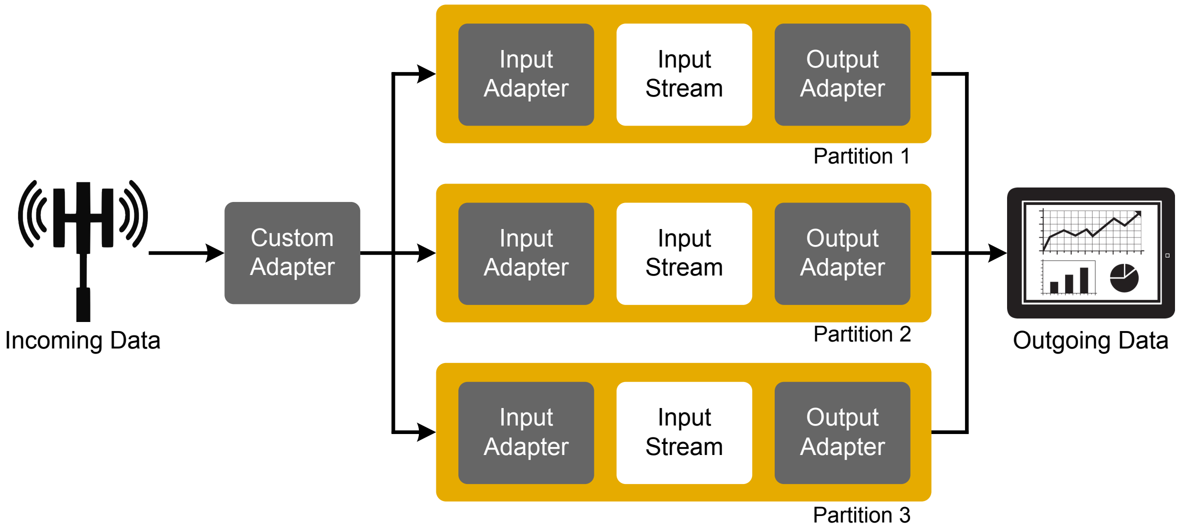


Figure 10: Partitioning Using a Custom Adapter

3. Partitioning Using a SPLITTER Statement

You can use the CCL SPLITTER object to subdivide input data based on specific criteria, and then a UNION statement to consolidate the data before sending it to the output adapter.

Advantages:

- More flexibility in the operations that you can perform on streams. For example, you first split the data, perform operations on the resulting streams, and then consolidate the data again.
- Allows for partitioning across cores.

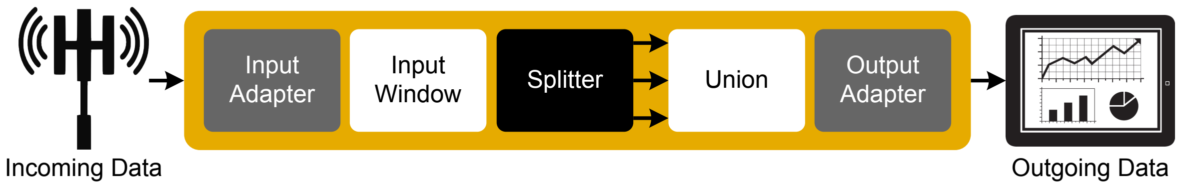


Figure 11: Partitioning Using SPLITTER and UNION Statements

Although the example in the illustration uses a single input adapter, you can use a SPLITTER when using multiple input adapters.

> **Note**
>
> Using the JOIN object does not provide the same performance benefit as using the UNION. In fact, the JOIN operation can degrade performance considerably, so to optimize performance, parallelizing your project using the SPLITTER/UNION combination is recommended over using JOIN.

In both the cases, the number of parallel instances is limited to the throughput of the union and the SPLITTER, when used. In addition, the number of parallel instances depends on the number of available CPUs.

4. Automatic Partitioning

You can create multiple parallel instances of a given element (keyed stream, stream, window, module) and partition input data into these instance. Partitioning data this way results in higher performance as the workload is split across the parallel instances. If using this scenario, you can partition the CCL elements using CUSTOM, HASH, or ROUND ROBIN partitioning.

Advantages:

- Ideal for complex projects which perform computationally expensive operations, such as aggregation and joins.
- Easy to add to a project.
- Allows for partitioning across cores.

Disadvantage:

- Lacks the ability to order output.

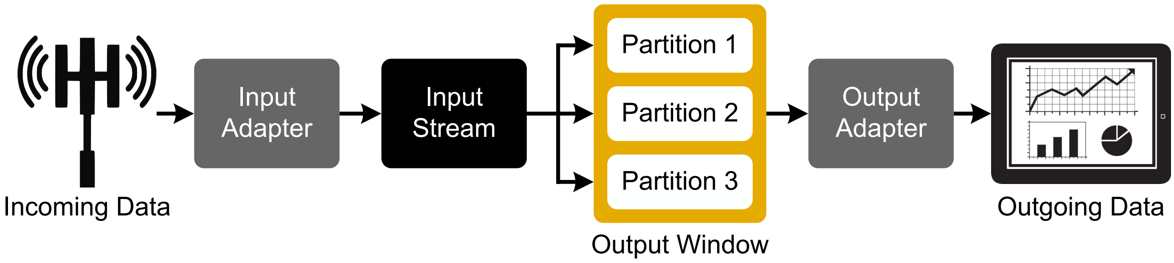


Figure 12: Automatic Partitioning

General Guidelines

Hash partitioning uses hash functions to partition data. The hash function determines which partition to place a row into based on the column names you specify as keys. These do not have to be primary keys. Round-robin partitioning distributes data evenly across partitions without any regard to the values.

Choose a type based on the calculations you are performing on the input data. For example, round-robin is sufficient for stateless operations like simple filters, but not for aggregation as this would produce differing results. Hash partitioning is necessary for grouping records together, but grouping may not evenly distribute the data across instances.

When implementing the scenarios above, use round-robin or key-based partitioning. Round-robin partitioning provides the most even distribution across the multiple parallel instances, but is recommended only for projects limited to insert operations (that is, no updates or deletes). For projects using insert, update, and delete operations, key-based partitioning is preferable. Any update or delete operation on a record should occur on the same path where the record was inserted, and only key-based partitioning can guarantee this. However, key-based partitioning can distribute load unevenly if the HASH function is not applied correctly, resulting in some partitions having a higher burden than others.

For more information on the SPLITTER and UNION statements, see the *SAP HANA Smart Data Streaming: CCL Reference*.

13.2 Distributing Load through Modularization

You can optimize performance by breaking projects into modules. This strategy spreads the load out to more cores, thereby increasing throughput.

Use modules to double, quadruple, and so on, the number of partitions, with very little additional code. The more partitions you create, the more you distribute the load.

For more information see the *Modularity* section in the *SAP HANA Smart Data Streaming: Developer Guide*, and the *SAP HANA Smart Data Streaming: CCL Reference*.

13.3 Streaming Data Flow

The throughput of the smart data streaming project depends on the throughput of the slowest component in the project.

Each stream in smart data streaming has an internal queue that holds up to 1024 messages; this queue is composed of multiple internal queues. The queue size is hard-coded and cannot be modified. An internal queue buffers data feeding a stream if that stream is unable to keep up with the inflowing data.

Consider an example where data flows from an input adapter, through streams A, B, and C, and then through an output adapter. If the destination target of the output adapter cannot handle the volume or frequency of messages being sent by the output adapter, the internal queue for the stream feeding the output destination fills up, and stream C cannot publish additional messages to it. As a result, the internal queue for stream C also fills up and stream B can no longer publish to it.

This continues up the chain until the input adapter can no longer publish messages to stream A. If, in the same example, the input adapter is slower than the other streams, messages will continue being published from stream to stream, but the throughput is constrained by the speed of the input adapter.

If your output destination is a database, you can batch the data for faster inserts and updates. Set the batch size for a database adapter in the data service for the cluster. For information on managing data services, see the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

Batching data carries some risk of data loss because the database adapters run on an in-memory system. To minimize the risk of data loss, set the batch size to 1.

13.4 Log Store Considerations

The size and location of your log stores can impact performance.

Sizing the log stores correctly is important. A store that is too small requires more frequent cleaning cycles, which severely degrades performance. In the worst case, the log store can overflow and cause the processing to stop. A store that is too large also causes performance issues due to the larger memory and disk footprint. For detailed information on calculating the optimal log store size, see *Sizing a Log Store* in the *SAP HANA Smart Data Streaming: Developer Guide*.

When storing smart data streaming data locally using log stores, use a high-speed storage device (for example, a raid array or SAN, preferably with a large dynamic RAM cache). Putting the backing files for log stores on single disk drives (whether SAS, SCSI, IDE, or SATA) always yields moderately low throughput.

13.5 Batch Processing

When stream processing logic is relatively light, inter-stream communication can become a bottleneck. To avoid such bottlenecks, you can publish data to the smart data streaming server in micro batches. Batching reduces the overhead of inter-stream communication and thus increases throughput at the expense of increased latency.

Smart data streaming supports two modes of batching: envelopes and transactions.

- Envelopes – When you publish data to the server using the envelope option, the server sends the complete block of records to the source stream. The source stream processes the complete block of records before forwarding the ensuing results to the dependent streams in the graph, which in turn process all the records before forwarding them to their dependent streams. In envelope mode, each record in the envelope is treated atomically, so a failure in one record does not impact the processing of the other records in the block.
- Transactions – When you publish data to the server using the transaction option, processing is similar to envelope mode in that the source stream processes all of the records in the transaction block before forwarding the results to its dependent streams in the data graph. Transaction mode is more efficient than envelope mode, but there are some important semantic differences between the two. The key difference between the two modes is that in transaction mode, if one record in the transaction block fails, then all records in the transaction block are rejected, and none of the computed results are forwarded downstream. Another difference is that in transaction mode, all resultant rows produced by a stream, regardless of which row in the transaction block produced them, are coalesced on the key field. Consequently, the number of resulting rows may be somewhat unexpected.

In both cases the number of records to place in a micro batch depends on the nature of the model and needs to be evaluated by trial and error. Typically, the best performance is achieved when using a few tens of rows per batch to a few thousand rows per batch. While increasing the number of rows per batch may increase throughput, it also increases latency.

13.6 Main Memory Usage

There are no SAP HANA smart data streaming configuration settings that directly set up or control RAM usage on the machine. However, a reference from smart data streaming counts records in the system, ensuring that only one copy of a record is present in memory, although multiple references to that record may exist in different streams.

Memory usage is directly proportional to the number of records in a project. To limit the amount of memory the entire instance of smart data streaming uses before it reports an out-of-memory condition, use the `ulimit` command to restrict the amount of memory available to each shell process.

13.7 Monitor Project Memory Usage

When the smart data streaming server is running at log level INFO and it is shut down cleanly, it reports the amount of memory consumed by various project components, adding this information to the smart data streaming project log file. You can also generate this report on-demand without shutting down.

The log level is a project configuration option on the **Advanced** tab of the Project Configuration editor in studio. A report detailing project statistics is printed if the level is set at 6 when the smart data streaming server shuts down.

The files generated by a project, including the project log file, are placed in the SAP HANA trace directory. Set the log level to 6 to generate a report when shutting down. To change the log level at run time, use the `streamingprojectclient` tool and execute:

```
streamingprojectclient -p [<host>:]<port></workspace-name/project-name> -c
<username>:<password> "loglevel 6"
```

Alternatively, use the `streamingprojectclient` command `get_memory_usage all` to generate this report without having to set the log level or shut the project down. The report measures input and output streams, queues, adapters, and reference caches for their memory use. Use this data to identify memory usage bottlenecks, and fine tune your project layout to improve throughput.

> **Note**
>
> In some cases when using `get_memory_usage all`, some rows of data that are shared by components are subsequently recorded more than once in the report. Rows in a gateway queue, for example, are likely to be in a store, and possibly an input queue as well. Cross-referencing between component reports can help identify where select rows are recorded, but the overall report nonetheless serves to highlight where excessive memory is being used.

When looking for component-specific data, qualify the component type and name in the project being tracked. Component types include:

- gateway
- stream/window
- global
- reference cache

Component names include:

- all
- `<client-ID><client-name>`
- `<stream-name>`
- `<adapter-name>`
- `<reference-name>`

> **Note**
>
> While running this report, you may notice a degradation in performance until the report is complete.

In this section:

13.7.1 Gateway Memory Usage

Gateway establishes queues where it stores incoming data and, depending on the type of subscription, creates additional queues, indices, and stores for aggregation. When requesting a report for gateway, you can request data for all client connections, specify a client IP address, or specify an IP address and name pairing.

Use the *streamingprojectclient* `get_memory_usage` command to view all gateway clients:

```
get_memory_usage gateway
```

Use the *streamingprojectclient* `get_memory_usage` command to view gateway clients on a specific IP:

```
get_memory_usage <IP-Address>
```

Use the *streamingprojectclient* `get_memory_usage` command to view one specifed client:

```
get_memory_usage <IP-Address><Client-ID>
```

A gateway report includes the following:

```
<Client-IP><Client-ID>:
Input Queue:
<subscriber-queue-memory-size> bytes in <subscriber-queue-number-of-records>
records
Aggregation Indices:
<aggregation-index-memory-size> bytes
Order by Store:
<order-by-memory-size> bytes in <number-of-records> records
Output Queue:
<output-queue-memory-size> bytes in <output-queue-number-of-rows> rows
```

The following example illustrates the report when using a specified client IP address and client ID:

```
[SP-6-131096] (71.084) sp(14723) Gateway Client 10.7.168.66 (129) Memory usage:
[SP-6-131094] (71.084) sp(14723) Queue: 0 bytes in 0 records
```

```
[SP-6-131097] (71.084) sp(14723) Aggregation Indices: 12,902 bytes in 26 records
[SP-6-131098] (71.084) sp(14723) Result Set Store: 1,846 bytes in 26 records
[SP-6-131095] (71.084) sp(14723) Gateway Output Store: 0 bytes in 0 rows
```

Considerations for gateway memory usage:

- Queues: You can adjust the settings of your connection tool to alleviate any strain on the connection. Increase the amount of records your tool can consume, or send fewer records by filtering data into other streams and subscribing to those streams. See the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for infomation on project configurations.
- Aggregation Indices: This is a temporary index that is only stored when specified in the query to the client connection. This memory usage ends when the query is finished.
- Result Set Store: Like aggregation indices, this memory usage is temporary.
- Output Store: This is a concern only if the subscription is pulsed, where records are collected and delivered at set intervals. If the output store is holding onto too much data, lower the pulse subscribe interval. See the *SAP HANA Smart Data Streaming: Utilities Guide* for information on controlling the pulse rate.

13.7.2 Stream and Window Memory Usage

When requesting a report for streams, request data for all streams in a project or a specific stream by name.

There are different kinds of streams to monitor. Most allocate a temporary amount of memory to process rows, and some streams keep rows and other data for the entire project lifecycle. Streams can also have CCLScript local declare block variables of basic data types, records, dictionaries, vectors, xml values, and event caches.

Use the *streamingprojectclient* `get_memory_usage` command to view the memory usage for all streams:

```
streamingprojectclient get_memory_usage stream
```

Use the *streamingprojectclient* `get_memory_usage` command to view memory usage for a single stream:

```
streamingprojectclient get_memory_usage stream '<stream-name>'
```

A report includes the following:

```
<Stream-Name>:
Store:
<store-memory-size> bytes in <number-of-stored-rows> records stored
Input Queue:
<Input-Queue-memory-size> bytes in <number-of-queued-rows> records
Transaction Export Queue:
<Transaction-Queue-memory-size> bytes in <number-of-queued-rows> records
Aggregation Index:
<Aggregation-Index-memory-size> bytes in aggregation index
Guaranteed Delivery Store:
<GD-Store-memory-size> bytes in <number-of-stored-rows> records
```

Considerations for stream or window memory usage:

- Store: A log store that is too large can hinder performance due to larger disk and memory requirements. To reduce memory usage, consider adjusting your retention policy on the store to retain less data .

> **Note**
>
> Log stores that are too small can cause processing to stop due to overflow. They can also cause significant performance degradation due to frequent cleaning cycles. See *Creating a Log Store* in the *SAP HANA Smart Data Streaming: Developer Guide* for information regarding log store optimization.

- Input Queue: Examine other objects that are consuming messages from the stream, and determine if their queues are full. If a queue is not full, check the object feeding into it for potential bottlenecks. Reduce the messages coming into the stream by filtering and distributing the data through multiple streams.
- Transaction Export Queue: Huge transactions are the primary cause of excessive memory use. Avoid constructing a transaction with an excessive number of records. Determine if any secondary objects like tables are joined into the row without any equality conditions.
- Aggregation Index: If a stream feeds input to an aggregation window directly, the memory usage of the aggregation index increases without bound. To prevent such unbounded growth, insert an intermediate window between the stream and the aggregation window. Consider lowering the retention policy of the store or unnamed window that has the aggregation..

> **Note**
>
> There are advantages and disadvantages to changing the retention policy. See *Aggregation* in the *SAP HANA Smart Data Streaming: Developer Guide* for more information.

- GD Mode: A stream with guaranteed delivery (GD) mode enabled stores records until all rows are fully processed. If GD Mode is enabled, but not required, disable it.

Depending on which CCLScript variables exist in the stream, a report includes:

```
CCLScript Variables:
<Stream-Dictionary-size> in bytes
<Stream-Event-Cache-size> in bytes
<Stream-Vector-size> in bytes
<Stream-Records-size> in bytes
<Stream-XML-Values-size> in bytes
<Primitive-Variables-size> in bytes
```

Considerations for CCLScript local declare block variables:

- Multiple variables of the same type will not be grouped together.
- The size of your variables are dependent on their usage. For more information on declare block variables, consult the *CCL Statements* section of the *SAP HANA Smart Data Streaming: CCL Reference*.

The following example illustrates a report when tracking streams for memory usage:

```
SP-6-124001] (191.065) sp(21115) Log store: 3,692 bytes in 52 records
[SP-6-124001] (191.065) sp(21115) Input queue: 700,000 bytes in 46,100 records
[SP-6-124001] (191.065) sp(21115) Export queue 1,000,000 bytes in 67,040 records
[SP-6-124001] (191.065) sp(21115) Aggregation Index: 1,545,000,000 bytes in
aggregation index
[SP-6-124001] (191.065) sp(21115) GD Store: 0 bytes in 0 records
[SP-6-124001] (191.065) sp(21115) StreamDic1: 3 bytes
[SP-6-124001] (191.065) sp(21115) StreamEvent1: 28,668 bytes
[SP-6-124001] (191.065) sp(21115) StreamVec1: 19 bytes
[SP-6-124001] (191.065) sp(21115) StreamVec2: 78 bytes
[SP-6-124001] (191.065) sp(21115) StreamRec1: 111 bytes
[SP-6-124001] (191.065) sp(21115) StreamRec2: 111 bytes
[SP-6-124001] (191.065) sp(21115) StreamXml1: 72 bytes
[SP-6-124001] (191.065) sp(21115) Primitive Variables: 32 bytes
```

Related Information

SAP HANA Smart Data Streaming: CCL Reference

13.7.3 CCLScript Variables Memory Usage

When requesting information on CCLScript memory usage, specify your search for either global or local declare blocks.

CCLScript variables include basic datatypes, records, dictionaries, vectors, xml values, and event caches. These variables are declared in DECLARE statements, which help to define a project's computations and logic. Local declare blocks are used in regular and Flex streams, while global declare blocks are available to an entire project.

A report for CCLScript variables can include the following:

```
<Stream-Dictionary-size> in bytes
<Stream-Event-Cache-size> in bytes
<Stream-Vector-size> in bytes
<Stream-Records-size> in bytes
<Stream-XML-Values-size> in bytes
<Primitive-Variables-size> in bytes
```

Use the following *streamingprojectclient* commands to report CCLScript variables present in local streams or a single stream:

```
streamingprojectclient get memory usage stream
```

```
streamingprojectclient get_memory_usage stream '<stream-name>'
```

The following example illustrates the report when tracking CCLScript variable memory use in streams:

```
[SP-6-124001] (191.065) sp(21115) streamDic1: 3 bytes
[SP-6-124001] (191.065) sp(21115) streamEvent1: 28,668 bytes
[SP-6-124001] (191.065) sp(21115) streamVec1: 19 bytes
[SP-6-124001] (191.065) sp(21115) streamVec2: 59 bytes
[SP-6-124001] (191.065) sp(21115) streamVec3: 13 bytes
[SP-6-124001] (191.065) sp(21115) streamRec1: 111 bytes
[SP-6-124001] (191.065) sp(21115) streamXml1: 72 bytes
[SP-6-124001] (191.065) sp(21115) Primitive Variables: 32 bytes
```

Note

When using `stream` and `stream <stream-name>` keywords for your report, other components of the stream are reported. See *Stream and Window Memory Usage*.

Use the *streamingprojectclient* command to report CCLScript variables in global declare blocks:

```
streamingprojectclient get_memory_usage global
```

The following example illustrates the report when tracking CCLScript variables' memory use in global declare blocks:

```
[SP-6-124001] (191.065) sp(21115) globalDic1: 64 bytes
[SP-6-124001] (191.065) sp(21115) globalRec1: 111 bytes
[SP-6-124001] (191.065) sp(21115) globalRec2: 311 bytes
[SP-6-124001] (191.065) sp(21115) globalRec3: 245 bytes
[SP-6-124001] (191.065) sp(21115) globalVec1: 66 bytes
[SP-6-124001] (191.065) sp(21115) globalVec2: 78 bytes
[SP-6-124001] (191.065) sp(21115) globalXml1: 72 bytes
[SP-6-124001] (191.065) sp(21115) Primitive variables: 32 bytes
```

Considerations for CCLScript declare block variables:

- Multiple variables of the same type are not grouped together.
- The size of your variables are dependent on their usage. For more information on declare block variables, consult the CCL Statements section of *SAP HANA Smart Data Streaming: CCL Reference*.

13.7.4 Reference Cache Memory Usage

References can store information in potentially unlimited-size caches. When requesting a report for memory allocated to the reference cache, specify the reference name.

Use the *streamingprojectclient* `get_memory_usage` command to view the size of a reference cache:

```
get_memory_usage <reference-name>
```

A report includes the following:

```
<reference-name>:
<cache-size> bytes in <number-of-cached-queries> queries
```

Considerations for reference memory usage:

- Change the retention policy on your reference by lowering the `maxCacheSize` parameter. This is the only way to reduce memory consumption for this component.

  > **i Note**
  >
  > There are significant advantages and disadvantages to adjusting this parameter. See *Reference Table Queries* in the *SAP HANA Smart Data Streaming: Developer Guide*.

13.8 CPU Usage

SAP HANA smart data streaming automatically distributes its processing load across all the available CPUs on the machine. If the processing of a data stream seems slow, monitor each stream's CPU utilization using either the `streamingmonitor` utility from the command line or through SAP HANA cockpit. If the monitoring tool shows one stream in the project using the CPU more than other streams, refine the project to ensure that the CPU is used evenly across the streams.

The queue depth is also a very important metric to monitor. Each stream is preceded by a queue of input records. All input to a given stream is placed in the input queue. If the stream processing logic cannot process the records as quickly as they arrive to the input queue, the input queue can grow to a maximum size of 1,024 records. At that point, the queue stops accepting new records, which results in the automatic throttling of input streams. Since throttled streams require no CPU time, all CPU resources are distributed to the streams with the full queues, in effect performing a CPU resource load balance of the running project. When a stream's input queue is blocked, but the stream has managed to clear half of the pending records, the queue is unblocked, and input streams can proceed to supply the stream with more data.

If this inherent load balancing is insufficient to clear the input queue for any given stream, the backup of the queue can percolate upward causing blockages all the way up the dependency graph to the source stream. If your monitoring indicates growing or full queues on any stream or arc of streams in the directed graph, examine this collection of streams to determine the cause of the slow processing.

13.9 TCP Buffer and Window Sizes

High throughput data transfers between clients and SAP HANA smart data streaming rely on the proper tuning of the underlying operating system's TCP networking system.

The data generated by clients for delivery to smart data streaming does not always arrive at a uniform rate. Sometimes the delivery of data is bursty. In order to accommodate large bursts of network data, large TCP buffers, and TCP send/receive windows are useful. They allow a certain amount of elasticity, so the operating system can temporarily handle the burst of data by quickly placing it in a buffer, before handing it off to smart data streaming for consumption.

If the TCP buffers are undersized, the client may see TCP blockages due to the advertised TCP window size going to zero as the TCP buffers on the smart data streaming server fill up. To avoid this scenario, tune the TCP buffers and window sizes on the server on which smart data streaming is running to between one and two times the maximum size that is in use on all client servers sending data to smart data streaming.

For information and best practices for determining and setting TCP buffer and window sizes, consult the documentation provided with your operating system.

13.10 Improving Aggregation Performance

Aggregation functions typically require the server to iterate over every element in a group. For this reason, the performance of the aggregation operator is inversely proportional to the size of the group.

Aggregation functions can be used in a SELECT statement along with a GROUP BY clause, or over event caches in CCLScript inside UDFs and Flex operators.

For the `sum()`, `count()`, `avg()`, and `valueInserted()` aggregation functions, the server can perform additive optimization, where the function executes in constant time. In such cases, the time it takes to perform an operation is the same regardless of group size.

In a SELECT statement, the server performs additive optimization provided functions eligible for optimization are used in all values being selected, with the exception of the columns referenced in the GROUP BY clause.

The following SELECT statement is optimized for additive optimization since all non-GROUP BY columns (`name`, `counter`, and `summary`) only use additive aggregation functions (that is, `valueInserted()`, `sum()`, and `count()`).

```
CREATE OUTPUT WINDOW AggResult
    SCHEMA (id INTEGER, name STRING, counter INTEGER, summary FLOAT)
    PRIMARY KEY DEDUCED
AS
    SELECT BaseInput.intData_1 AS id,
        valueInserted(BaseInput.strData_1) AS name,
        count(BaseInput.intData_1) AS counter,
        sum(BaseInput.dblData_1) AS summary
FROM BaseInput
GROUP BY BaseInput.intData_1;
```

> **Note**
>
> For optimal peformance, when selecting only the column in a SELECT statement with a GROUP BY clause, use the valueInserted function, where feasible.

The following SELECT statement is not optimized for additive optimization since one of the non-GROUP BY columns (`name`) directly selects a column which cannot be computed additively.

```
CREATE OUTPUT WINDOW AggResult
    SCHEMA (id INTEGER, name STRING, counter INTEGER, summary FLOAT)
    PRIMARY KEY DEDUCED
AS
    SELECT BaseInput.intData_1 AS id,
        BaseInput.strData_1 AS name,
        count(BaseInput.intData_1) AS counter,
        sum(BaseInput.dblData_1) AS summary
FROM BaseInput
GROUP BY BaseInput.intData_1;
```

When applying aggregation functions over an event cache, additive optimization is turned on when using the `sum()`, `count()`, `avg()`, or `valueInserted()` functions only in the ON clause of a Flex operator. The additive optimization does not apply when functions are used inside a UDF.

The following Flex stream computes the sum in the ON clause additively, since the `sum()` function is computed additively and the used EventCaches (e0,e1) are declared locally.

```
CREATE INPUT WINDOW In1
    SCHEMA (c1 INTEGER, c2 STRING, c3 INTEGER, summary FLOAT)
    PRIMARY KEY (c1, c2);
CREATE FLEX MyFlex
    IN In1
    OUT OUTPUT WINDOW FlexOut
    SCHEMA (c1 INTEGER, c2 INTEGER, c3 INTEGER, c4 INTEGER)
    PRIMARY KEY (c1, c2)
BEGIN
    declare
        eventCache(In1, coalesce) e0;
        eventCache(In1, coalesce) e1;
    end;
    ON In1 {
     {
        output setOpcode([c1=In1.c1;c2=In1.c2;|
c3=sum(e0.c1);c4=sum(e1.c3);],getOpcode(In1));
     }
    };
```

```
END;
```

The following Flex stream is not computed additively, since the `stddev()` function cannot be computed additively.

```
CREATE INPUT WINDOW In1
    SCHEMA (c1 INTEGER, c2 STRING, c3 INTEGER)
    PRIMARY KEY (c1, c2);
CREATE FLEX MyFlex
    IN In1
    OUT OUTPUT WINDOW FlexOut
    SCHEMA (c1 INTEGER, c2 INTEGER, c3 INTEGER, c4 FLOAT)
    PRIMARY KEY (c1, c2)
BEGIN
    declare
        eventCache(In1, coalesce) e0;
        eventCache(In1, coalesce) e1;
    end;
    ON In1 {
     {
        output setOpcode([c1=In1.c1;c2=In1.c2;|
c3=sum(e0.c1);c4=stddev(e1.c3);],getOpcode(In1));
     }
    };
END;
```

Another restriction is that additive optimizations are disabled when functions are used inside nonlinear statements (IF, WHILE, FOR, and CASE statements). To enable additive optimizations when using a function within a nonlinear statement, assign the result of the function to a variable outside of the statement. Then use the variable inside the nonlinear statement.

Note

The function used within the nonlinear statement must be from the set of functions eligible for additive optimization.

The following SELECT statement is not optimized for additive optimization since one of the expressions (CASE) in the SELECT list is a nonlinear expression:

```
CREATE OUTPUT WINDOW AggResult
    SCHEMA (id INTEGER, name STRING, counter INTEGER, summary FLOAT)
    PRIMARY KEY DEDUCED
AS
    SELECT BaseInput.intData_1 AS id,
        valueInserted(BaseInput.strData_1) AS name,
        CASE WHEN (count(BaseInput.intDATA_1) < 100) THEN 0 ELSE 1 END AS
counter,
        sum(BaseInput.dblData_1) AS summary
FROM BaseInput
GROUP BY BaseInput.intData_1;
```

13.11 Switch from Decimal to Money Datatype

Using the `money` datatype rather than the `decimal` datatype can improve the performance of a project.

The `money` datatype cannot handle the full range of values that the `decimal` datatype can. Ensure that you do not expect any values outside of the `money` datatype's range before making this change.

13.12 Recompiling Streaming Projects

Recompile existing streaming projects that were compiled with earlier compiler versions to take advantage of the latest enhancements.

Context

Enhancements to the compiler are made constantly to improve the performance of SAP HANA smart data streaming projects.

Procedure

1. Back up the existing `.ccx` file for the project. By default, these files are in the `%STREAMING_HOME%\bin` folder on Windows machines and the `$STREAMING_HOME/bin` directory on Linux machines.
2. Open the SAP HANA Streaming Run-Test perspective, and compile the streaming project. Refer to *Compiling a Streaming Project* in the *SAP HANA Smart Data Streaming: Developer Guide* for details.

14 Appendix: Recommendations for Project Portability

Guidelines on what you can do to move projects from one smart data streaming installation to another without having to edit your CCL to work with the new environment, and therefore, avoid having to recompile your CCL.

Log Stores

For any log stores in your project, create a custom parameter and make its value the file name and path of your log store file. To change this value later, you can do so in the project configuration (ccr) file without having to make changes to your CCL and recompile. For example, if you wanted to create a parameter called `LogStoreFilepath`:

```
DECLARE
PARAMETER string LogStoreFilepath := '/<folder>/<folder>/<filename>'
END;
```

Then in the CREATE LOG STORE statement, specify your custom parameter name as the value for the `file name` property. For example:

```
CREATE LOG STORE storename
PROPERTIES
filename=LogStoreFilepath;
```

Adapter Property Sets

For any adapter properties that require file path values, use property sets instead of specifying values directly in CCL. This way, you can edit the property values in the project configuration (ccr) file instead of editing your CCL and having to recompile again.

Also, use an environment variable to specify the base directory of the adapter property file path. .

Here is an example of a property set that uses an environment variable called `FOO`:

CCL:

```
ATTACH INPUT ADAPTER csvInputAdapter
TYPE toolkit_file_csv_input
TO NEWSTREAM
PROPERTIES propertyset = 'CSV File Input';
```

CCR:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://www.myserver.com/streaming/project_config/2010/08/">
  <Runtime>
```

```
<AdaptersPropertySet>
    <PropertySet name="CSV File Input">
    <Property name="dir">${FOO}/Examples/exampledata/</Property>
    <Property name="file">csv_input.csv</Property>
  </PropertySet>
```

15 Appendix: Troubleshooting

Common techniques for troubleshooting issues you may encounter in studio.

To help troubleshoot events and view studio activity, refer to the studio log file. The studio log file resides in your workspace directory under `workspace/.metadata/.log`.

In this section:

Issue: When starting an adapter without editing its sample XML configuration file, the adapter fails to start.

A Studio Project Does Not Run, Reports Login Failure [page 356]

Issue: You are unable to run a project in a cluster and errors display.

Machine Learning Model Reads Rows as Null [page 357]

Issue: When running a machine-learning model, all rows in a column register as null values.

Error Compiling CCL in PowerDesigner [page 357]

Issue: You receive the following error when importing CCL into a Streaming Schema model in PowerDesigner:

Cannot Add a Server Connection in Studio [page 357]

Issue: When working in studio, you cannot add a server connection through the SAP HANA Streaming Run-Test perspective server view.

15.1 Cannot Connect to Server

Issue: When trying to run a project, studio returns a connection error.

When running a project, the following error message appears:

```
<Cannot connect to server>
```

This may be due to an invalid license key.

Solution: View the studio log file.

1. Select **Help > About Studio**.
2. Click **Configuration Details**.
3. Click **Configuration**.
4. Click **View Error Log**.
5. If prompted, select a text editor to view the file.
6. Locate the error. If the log file entry indicates a problem with your license, refer to the information on licensing in the *SAP HANA Smart Data Streaming: Installation and Update Guide* or the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide
SAP HANA Smart Data Streaming: Installation and Update Guide

15.2 Cannot Connect to the Cluster

Issue: When running a project, you cannot connect to the SAP HANA smart data streaming cluster.

Solution: Check the status of the SAP HANA smart data streaming server, and restart if necessary.

1. Verify that the smart data streaming host is running. See *Verify that the Streaming Service is Running* in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for detailed instructions.
2. If the host is not running, start it. See *Start the Streaming Service* in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide* for detailed instructions.
3. If the host cannot be started, check the smart data streaming log files in the SAP HANA trace directory for errors.
4. Verify that the SAP HANA index server is running.

> **Note**
>
> A common cause for issues with starting a smart data streaming host is the SAP HANA index server not running.

15.3 Cannot Open a Project in Studio

Issue: Attempts to open a project fail and cause errors.

You encounter the following error:

```
Could not open the editor: Invalid thread access
```

This issue is caused by conflicts with previous environment variable settings, or by missing environment variables.

Solution: Set environment variables and start studio using the command prompt.

1. From the command line, navigate to the smart data streaming install directory. The default name of this directory is `/Streaming`.
2. Set the environment variables:

Operating System	Step
Windows	Run `STREAMING.bat`
UNIX	Source `STREAMING.csh`, `STREAMING.sh`, or `STREAMING.env`

3. Navigate to `<install-dir>/STREAMING-1_0/studio`.
4. Run `streamingstudio.exe` to start SAP HANA studio.

15.4 Schema Discovery Fails

Issue: When executing discovery for an element in a streaming project, discovery fails.

Discovery uses the Default Server URL preference to retrieve the list of available data services. The default server URL is esp://localhost:9786. Because there is no localhost smart data streaming server, no data services are located.

Solution: Change the Default Server URL preference to an available smart data streaming server.

1. In the SAP HANA Streaming Run-Test perspective, select **Window** > **Preferences**.
2. In the Preferences dialog, select **SAP HANA smart data streaming**.
3. In the **Default Server URL** field, click **Change** and select a server from the Select Default Server URL dialog. Click **OK**.
4. In the Preferences dialog, click **Apply**, then **OK**.

15.5 Changing Date Format for Playback

Issue: Unable to load or playback files containing dates in formats other than UTC.

The studio playback tool assumes all dates and times are in UTC format.

Solution: Set a date mask to change the order of date and time values.

1. Open the SAP HANA Streaming Run-Test perspective.
2. In the Playback view, enter a date mask in the **XML/CSV datemask** field using the following format: `%Y-%m-%dT%H:%M:%S`
   Change the value order as necessary. See *Playback View* in the *SAP HANA Smart Data Streaming: Developer Guide* for more information.

You cannot change delimiters from within studio. To learn how to specify a different delimiter from the command line, see *Server Executables* in the *SAP HANA Smart Data Streaming: Utilities Guide*.

Related Information

SAP HANA Smart Data Streaming: Utilities Guide

15.6 An External Adapter Fails to Start

Issue: Your attempts to start an external adapter fail.

When attempting to run an external adapter an error message similar to the following appears:

```
Failed call to:https://<Streaming-hostname>:61308/RPC2 (Failed to read server's
response: <Streaming-hostname>) java.io.IOException: Failed call to:https://
<Streaming-hostname>:61308/RPC2 (Failed to read server's response: <Streaming-
hostname>)
```

This error is an example of the smart data streaming server not being resolved.

Solution: Use the `ping` command to verify that the hostname of the server to which you are trying to connect can be resolved. If the hostname cannot be resolved:

1. Determine the IP address of the host on which the smart data streaming server is running. Run this command from that machine:

   ```
   nslookup <hostname>
   ```

2. Add the following line to `C:\Windows\System32\drivers\etc\hosts` (Windows) or `/etc/hosts` (UNIX):

   ```
   <ip-address-of-server-hostname>        <Server-hostname>
   ```

15.7 External Managed Adapters Do Not Start in Desired Order

Issue: When using multiple external managed adapters in a project, you cannot implicitly guarantee that they will start in the desired order.

Solution: To enforce a specific ordering of external managed adapters:

1. In your CCL project, use the ADAPTER START statement with the NOSTART clause to prevent the adapters from starting with the project. For more information and an example of usage, see the *ADAPTER START Statement* topic in the *SAP HANA Smart Data Streaming: CCL Reference*.
2. Use the `streamingprojectclient` utility to start the adapters in your desired order. You can use a script to automate the command calls, and add short delays as necessary to ensure that all adapters are called and started in your desired order.

15.8 Playback is Too Fast or Slow

Issue While using the playback tool in the SAP HANA Streaming Run-Test perspective, data plays back too fast or too slow.

By default, studio runs projects in Full Rate mode. This playback mode is dependent on external factors, such as the computer running studio.

Solution: Change the playback mode and adjust the playback rate.

1. Open the SAP HANA Streaming Run-Test perspective.
2. In the Playback view, select the **rec/ms** playback mode.
3. Set your preferred speed in the **rec/ms** field. For example, set your speed to 0.01 to run the project at a speed of 0.01 records per millisecond.
4. Run your project. If necessary, adjust the speed setting using the **At timestamp rate** slider. Using the slider allows you to change playback speed while the project is running, since the **rec/ms** setting can only be directly changed while a project is stopped.

15.9 Studio Crashes and You Are Unable to Restart It

Issue: The studio workspace may have become corrupt.

Solution: Migrate your projects to a new workspace.

1. Rename the corrupt workspace by appending the string `_OLD` to its name.
2. Shut down and restart studio. Upon restarting, studio creates a new workspace folder.
3. In the SAP HANA Streaming Development perspective, import your projects into the new workspace. See *Importing a Streaming Project into SAP HANA Studio* in the *SAP HANA Smart Data Streaming: Developer Guide* for detailed steps.

15.10 Retention Policy Errors

Issue: In a project, windows without retention policies cause errors, and the project cannot compile or run.

When compiling or running a project, any window without a retention policy causes one or both of the following error messages:

- `159211-4_WINDOW_NO_RETENTION_POLICY`
- `The Cluster has encountered a project error. Check the Cluster policies for this project. Cluster error message: null.`

The first message contains a generic resource number which indicates that the installer did not set the STREAMING_HOME environment variable correctly.

Solution: To fix this issue:

1. Ensure that STREAMING_HOME is set as a system variable.
   1. Go to **Control Panel**, then select ▸ **System** ▸ **Advanced System Settings** ▸ **Environment Variables** ◂.
   2. Set the STREAMING_HOME variable to `<install-dir>\STREAMING-1_0`.
2. Restart studio.

15.11 Stream View Causes Project to Run Slower

Issue: A project runs slower when streams or windows are open in the Stream View.

As studio tries to keep up with the data flowing through the server, it blocks incoming data and slows down the project.

Solution: Enable lossy subcription. This option allows the studio subscription to lose data if it is unable to keep up with the data flowing through the server.

1. Go to ▸ **Window** ▸ **Preferences** ▸ **SAP HANA smart data streaming** ▸ **Run Test** ◂.
2. Select **Streamviewer lossy subscription**. Click **OK** to save.

15.12 Stream View Displays Partial Data

Issue: In the SAP HANA Streaming Run-Test perspective, the Stream View does not show all rows of a running project.

Solution: Increase the number of visible rows in the Stream View preferences.

> **Note**
>
> Increasing the number of rows will also increase memory usage in studio.

1. Open the SAP HANA Streaming Run-Test perspective.
2. In the top right corner of the Stream View, select **Set StreamViewer Number of Rows Displayed.** .
3. Enter a new number of rows, then click **OK** to save.
4. (Optional) If your project has stopped running, re-run the project to see the new number of rows.

For more information on Stream View preferences, see the *Viewing a Stream* topic in the *SAP HANA Smart Data Streaming: Developer Guide*.

If Stream View does not show any data, the data may have already been processed. See Stream View Displays No Data [page 356] for more information.

15.13 Stream View Displays No Data

Issue: When running a project that uses one or more file input adapters, Stream View does not show any data.

File input adapters read data into projects at very high speeds. Streams are stateless and do not store any data, so by the time a user opens a stream in the viewer, all the input data has already been processed. Since the Stream View can only show new events since the time it was opened, there is nothing to see.

Solution: Manually load the data using the File Upload tool.

1. In the SAP HANA Streaming Development perspective, remove the ATTACH ADAPTER statement from the CCL, or delete the adapter using the visual editor.
2. Compile and run the project.
3. In the SAP HANA Streaming Run-Test perspective, open the necessary streams in Stream View.
4. Open the File Upload tool to manually load the data. For more information, see the *Uploading Data to the SAP HANA Smart Data Streaming Server* topic in the *SAP HANA Smart Data Streaming: Developer Guide*.

If Stream View shows some data but the rest is missing, adjust your preferences to increase the number of visible rows. See Stream View Displays Partial Data [page 355] for more information.

15.14 An Adapter Fails to Connect to a Project

Issue: When starting an adapter without editing its sample XML configuration file, the adapter fails to start.

The adapter may be unable to connect to the example workspace specified in the sample XML adapter configuration file if the URI specified in the file uses `esp` instead of `esps`. This causes the adapter to fail to connect because SSL is enabled by default on SAP HANA smart data streaming and cannot be disabled.

Solution:

1. Ensure the URI in the adapter XML configuration file uses `esps` not `esp`.
2. If using one of the adapter examples provided with your installation, edit the `set_example_env.bat` or `set_example_env.sh` script file to specify:

```
set ADAPTER_EXAMPLE_CLUSTER_NODE_PROTOCOL=esps
```

15.15 A Studio Project Does Not Run, Reports Login Failure

Issue: You are unable to run a project in a cluster and errors display.

```
Failed to connect to server "esps://<host>:30026".
Reason: "Failed to login server"
```

Studio reports these errors when it has an SSL mismatch with the smart data streaming server. For example, since SSL is enabled on the server by default and cannot be disabled, the mismatch results from the studio connection definition for that server not specifying SSL.

Solution: Correct the connection definition in studio. For details, see the *Configuring a Cluster Connection* topic in the *SAP HANA Smart Data Streaming: Configuration and Administration Guide*.

15.16 Machine Learning Model Reads Rows as Null

Issue: When running a machine-learning model, all rows in a column register as null values.

Solution: Model column titles are case-sensitive. Ensure that the schema of the input stream matches the schema saved in the model.

1. In the studio visual editor, expand the input stream view to see schema details, or open the CCL editor for text view.
2. From the data services view, open the model definition.
3. Compare the schema of the stream with the model input schema, and fix any inconsistencies.

Avoid any deliberate null values in the input data, as PAL machine-learning functions cannot infer null values. Replace any null values with default values using the SQL console.

To make sure that the model function is picking up all events, subscribe to the input stream with another stream or window. This way, you can keep track of any missing entries in the data set.

15.17 Error Compiling CCL in PowerDesigner

Issue: You receive the following error when importing CCL into a Streaming Schema model in PowerDesigner:

```
There was an error in compiling the given CCL. Please check the log file.
```

This issue occurs on Windows 2008 R2 if PowerDesigner is installed in the system program folder. The default permission for this folder is ReadOnly, which prevents PowerDesigner from writing to this location.

Solution: Add the user Everyone to the PowerDesigner installation folder and grant this user read and write permissions.

15.18 Cannot Add a Server Connection in Studio

Issue: When working in studio, you cannot add a server connection through the SAP HANA Streaming Run-Test perspective server view.

When adding a server connection, you receive the following error:

```
Invalid Cluster Manager Field, enter a valid Cluster Manager Value.
```

Solution: This error occurs when the value in the **Host Name** field is entered incorrectly.

To fix this issue, enter a valid host URL in the **Host Name** field, without the `esp` or `esps` prefix. For example, `myhostname`, but not `esps://myhostname`.

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of wilful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).

go.sap.com/registration/contact.html