# Table of Contents

1 **Introduction to SAP Event Stream Processor** ................................................. 6  
1.1 Events ........................................................................................................ 7  
1.2 Event Stream Processor Compared to Databases ............................................. 8  
1.3 Data-Flow Programming ............................................................................. 8  
1.4 Streams, Windows, Adapters, and Continuous Queries in Projects ..................... 10  
1.5 Streams Versus Windows ............................................................................. 11  
1.6 Getting Results from a Project .................................................................... 12  
1.7 Schemas ...................................................................................................... 12  
1.8 Operation Codes ......................................................................................... 12  
1.9 Event Stream Processor Components ............................................................. 13  
1.10 Input and Output Adapters ......................................................................... 14  
1.10.1 Custom Adapters .................................................................................... 15  
1.11 Authoring Methods .................................................................................... 15  
1.12 Continuous Computation Language ............................................................. 16  
1.13 CCLScript ................................................................................................... 17  

2 **Cluster Connectivity in Studio** ................................................................. 18  
2.1 Changing Networking Preferences ................................................................. 18  
2.2 Running a Project from a Local Cluster ......................................................... 19  
2.3 Configuring the Local Cluster for a Remote Connection ................................. 20  
2.4 Connecting to a Remote Cluster ................................................................... 20  
2.5 Connecting to a Kerberos-Enabled Server ..................................................... 21  
2.6 Connecting to an RSA-Enabled Server .......................................................... 22  
2.7 Configuring a Remote Cluster Connection .................................................... 23  
2.8 Modifying a Remote Cluster Connection ....................................................... 24  
2.9 Managing Secure Storage Credentials for an ESP Server Connection ............. 25  
2.9.1 Storing Credentials in Secure Storage for an ESP Server Connection .......... 25  
2.9.2 Modifying Credentials in Secure Storage for an ESP Server Connection .... 26  

3 **Getting Started with a Project** ................................................................. 27  
3.1 Launching Studio .......................................................................................... 27  
3.2 Studio Basics ............................................................................................... 27  
3.2.1 Project Explorer ...................................................................................... 29  
3.3 The Studio Log File ..................................................................................... 29  
3.4 Creating a Project ....................................................................................... 30  
3.4.1 Opening a Project in the Workspace ......................................................... 31  
3.4.2 Importing an Existing Project ................................................................ 32  
3.4.3 Importing Multiple Projects ................................................................... 33  
3.5 Renaming a Project ..................................................................................... 33
10.7.2 Playing Recorded Data ................................................... 161
10.8 Debugging ........................................................................ 161
  10.8.1 Event Tracer View .......................................................... 162
  10.8.2 Debugging with Breakpoints and Watch Variables ............. 164

11 Customizing the Studio Work Environment ................................. 171
  11.1 Editing Studio Preferences ................................................... 171
  11.2 Manual Input Settings ........................................................... 172
  11.3 Rearranging Views in a Perspective ....................................... 174

12 Troubleshooting ...................................................................... 175
  12.1 A Studio Project Does Not Run, Reports Login Failure ............. 175
  12.2 Cannot Publish or Subscribe to a Project that Seems to be Running 175
  12.3 A Legacy Project Fails to Compile or Run ............................... 176
  12.4 Cannot Connect to Server ....................................................... 176
  12.5 Cannot Connect to the Cluster ............................................... 177
  12.6 Cannot Open a Project in Studio ............................................. 178
  12.7 Changing Date Format for Playback ...................................... 178
  12.8 An External Adapter Fails to Start ......................................... 179
  12.9 Prompted for Local Cluster Password ................................... 180
  12.10 Playback is Too Fast or Slow ............................................... 180
  12.11 Studio Crashes and You Are Unable to Restart It .................. 181
  12.12 Retention Policy Errors ....................................................... 181
  12.13 Stream View Causes Project to Run Slower ......................... 182
  12.14 Stream View Displays Partial Data ....................................... 183
  12.15 Stream View Does Not Display Any Data ............................ 183
  12.16 An Adapter Fails to Connect to a Project .............................. 184
  12.17 An External Adapter Fails to Start ........................................ 185
1 Introduction to SAP Event Stream Processor

SAP® Event Stream Processor enables you to create and run your own complex event processing applications to derive continuous intelligence from streaming event data in real time. Using raw data from external sources, Event Stream Processor can identify the actionable pieces of information out of the flood of data, and redeposit the relevant information wherever necessary, such as an SAP HANA table.

SAP Event Stream Processor and Complex Event Processing

SAP Event Stream Processor is a form of complex event processing (CEP), 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 Event Stream Processor, you can analyze events as they stream in and identify emerging threats and opportunities as they happen. The ESP server filters, aggregates, and summarizes data to enable better decision making based on more complete and timely information.

ESP is not an end-user application, but an enabling technology that provides tools that make it easy to develop and deploy both simple and complex projects. It provides a highly scalable runtime environment in which to deploy those projects.

SAP Event Stream Processor provides high-level tools for defining how events are processed and analyzed. Developers can work in either a visual or text-oriented authoring environment. You can define logic that is applied to incoming events to:

- Combine data from multiple sources, producing derived event streams that include richer and more complete information.
- Compute value-added information to enable rapid decision making.
- Watch for specific conditions or patterns to enable instantaneous response.
- Produce high-level information, such as summary data, statistics, and trends to see the big picture, or the net effect, of many individual events.
- Continuously recompute key operating values based on complex analysis of incoming data.
- Collect raw and result data into a historical database for historical analysis and compliance.

SAP Event Stream Processor Runtime Environment

As an engine for an event-driven architecture (EDA), Event Stream Processor can absorb, aggregate, correlate, and analyze events to produce new high-level events that can trigger responses, and high-level information that shows the current state of the business. SAP Event Stream Processor:

- Processes data continuously as it arrives.
- Processes data before it is stored, thus achieving extremely high throughput and low latency, enabling better decision making based on more complete and timely information.
- Separates business logic from data management, making it easier to maintain the business logic and reducing total cost of ownership.
- Provides enterprise class scalability, reliability, and security.
1.1 Events

A business event is a message that contains information about an actual business event that occurred. Many business systems produce streams of such events as things happen.

With SAP Event Stream Processor, through SAP Event Stream Processor Studio or CCL, you can use streams, windows, and keyed streams with adapters to create complex projects. Streams, windows, and keyed streams allow you to consume and process input events and generate output events.

**Note**
Using CCL, you can also include delta streams in your projects. SAP recommends delta streams for advanced users only.

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

- Financial market data feeds that transmit trade and quote events, where each event may consist of ticket symbol, price, quantity, time, and so on.
- Radio Frequency Identification System (RFID) sensors that transmit events indicating that an RFID tag was sensed nearby.
- Electronic sensors that 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, button, or control on a Web site.
- Database transaction events, which occur each time a record is added to a database or updated in a database.

**Event Blocks**

Business events can be published into ESP projects in collections called Event Blocks, improving the performance of your ESP projects. Event blocks come in two different types: envelopes and transactions. As an event block is being processed by a window, resulting rows are not sent downstream immediately. Instead, they are stored until the last event of the block is processed, and the resulting events are then 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 will be discarded if any one event in the block fails. This behavior can be used 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.
1.2 Event Stream Processor Compared to Databases

ESP complements traditional databases to help solve new classes of problems where continuous, event-driven data analysis is required.

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

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

ESP is not an in-memory database, although it stores all data in memory. ESP is optimized for continuous queries, rather than on-demand queries and transaction processing, as shown in the figure.

1.3 Data-Flow Programming

SAP® Event Stream Processor uses data-flow programming for processing event streams.

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 sources to outputs.

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 the figure shows a simple example of this.

Each of the continuous queries in this simple 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. 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.
Table 1: Data-Flow Diagram Contents

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PriceFeed</td>
<td>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. The difference is that in Event Stream Processor, the streaming data is not stored in a database.</td>
</tr>
<tr>
<td>Positions</td>
<td>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 aggregated.</td>
</tr>
<tr>
<td>VWAP</td>
<td>Represents a simple continuous query that performs an aggregation, similar to a SQL Select statement with a GROUP BY clause.</td>
</tr>
</tbody>
</table>

Figure 2: Data-Flow Programming - Simple Example
### 1.4 Streams, Windows, Adapters, and Continuous Queries in Projects

ESP projects are like applications, and consist of a set of event streams, any other required datasources, and the business logic applied to incoming event data to produce results.

At its most basic level, a project consists of:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input streams and windows</strong></td>
<td>Receives input data flowing into the project. An input stream can receive incoming event data on an event-driven basis, and can also receive 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.</td>
</tr>
<tr>
<td><strong>Adapters</strong></td>
<td>Connects an input stream or window to a datasource. ESP includes a large set of built-in adapters as well as an SDK that you can use to build custom adapters. Adapters can also connect an output stream or window to a destination. While an adapter connects the project to external inputs and outputs, technically it is not part of the project.</td>
</tr>
</tbody>
</table>
# 1.5 Streams Versus Windows

Both streams and windows process events. The difference is that windows have state, meaning they can retain and store data, while streams are stateless and cannot.

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

By contrast, a window consists of a table where incoming events can add rows, update existing rows, or delete rows. You can set the size of the window based on time, or on the number of events recorded. For example, a window might retain all events over 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 must be 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 for performing aggregate operations, as this cannot be done on streams.

## Input, Output, and Local Streams and Windows

Streams and windows can be designated as input, output, or local. 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.

Local and output streams and windows take their input from other streams or windows, rather than from adapters, and they apply a continuous query to produce their output. Local streams and windows are identical to output streams and windows, except that local streams and windows are hidden from outside subscribers. Thus, when a subscriber selects which stream or window to subscribe to, only output streams and windows are available.

**Note**

The palette view in the visual editor lists local and output streams as derived streams, and lists local and output windows as derived windows.
1.6 Getting Results from a Project

SAP Event Stream Processor has four ways to get output from a running project.

- Applications receive information automatically from output adapters attached to a stream when you build the project. For example, you can load data into an SAP HANA database using an output adapter.
- Applications can subscribe to data streams by means of an external subscriber, which users can create using subscription APIs provided with the product.
- You can start a new project that binds (connects) to a stream in a running project, without reconfiguring the project.
- You can run on-demand queries against output windows in a running ESP project. This is similar to querying a database table.
  - From the command line, using the `streamingquery` tool. For more information, see the SAP Event Stream Processor: Utilities Guide.
  - In Studio, using the SQL Query view tools.
  - From third-party applications, using the ODBC Driver for ESP. For more information, see the SAP Event Stream Processor: Utilities Guide.

1.7 Schemas

Each stream or window has a schema, which defines the columns in the events produced by the stream or window.

Each column has a name and datatype. All events that output from a single stream or window have an identical set of columns. For example:

- An input stream called RFIDRaw, coming out of an RFID reader, may have columns for a ReaderID and a TagID, both containing string data.
- An input stream called Trades, coming from a stock exchange, may have columns for the Symbol (string), Volume (integer), Price (float), and Time (datetime).

1.8 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 Event Stream Processor 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. ESP needs to maintain a current view of the set of information as the incoming events continuously update it. Two common examples are order books for
securities in capital markets, and open orders in a fulfillment system. In both applications, incoming events may indicate the need to:

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

To handle information sets that are updated by incoming events, Event Stream Processor recognizes the following opcodes in incoming event records:

- **insert** Insert the event record.
- **update** Update the record with the specified key. If no such record exists, it is a runtime error.
- **delete** Delete the record with the specified key. If no such record exists, it is a runtime error.
- **upsert** If a record with a matching key exists, update it. If a record with a matching key does not exist, insert this record.
- **safedelete** If a record with a matching key exists, delete it. If a record with a matching key does not exist, do nothing.

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

Refer to the SAP Event Stream Processor: Developer Guide for details on how windows and streams interpret the opcodes on incoming event records and generate opcodes for output records.

### 1.9 Event Stream Processor Components

SAP Event Stream Processor includes a server component for processing and correlating streams of data, a studio environment for developing, testing, and starting applications that run on the server, and administrative tools.

Components include:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP server</td>
<td>The software that processes and correlates data streams at runtime. SAP Event Stream Processor can process and analyze hundreds of thousands of messages per second. Clustering provides scale-out support to the server. A server cluster lets users run multiple projects simultaneously, provides high availability and failover, and lets you apply centralized security and support for managing cluster connections.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESP Studio</td>
<td>An integrated development environment for creating, modifying, and testing SAP ESP projects. You can access it directly from SAP HANA studio by installing the SAP Event Stream Processor Studio plugin for SAP HANA studio, or you can run it as a standalone application.</td>
</tr>
<tr>
<td>CCL compiler</td>
<td>The compiler that translates and optimizes projects for processing by the ESP server. It is invoked by the Studio or from the command line.</td>
</tr>
<tr>
<td>Input and output adapters</td>
<td>The components that establish connections between ESP and datasources, as well as the connections between the ESP server and the consumers that will receive output from SAP Event Stream Processor.</td>
</tr>
<tr>
<td>Integration SDK</td>
<td>A set of APIs for creating custom adapters in C/C++, Java, and .NET; for integrating custom function libraries; and for managing and monitoring live projects.</td>
</tr>
<tr>
<td>Utilities</td>
<td>A set of executables that offer command line access to many administrative, project development, publishing and subscription, and other features.</td>
</tr>
<tr>
<td>SAP ESP Cockpit</td>
<td>A Web-based tool for managing ESP cluster nodes, projects, adapters, and other components. For more information, see the <em>SAP Event Stream Processor: Cockpit Guide</em>.</td>
</tr>
</tbody>
</table>

Do not mix components from different versions of Event Stream Processor. For example, do not run the ESP server from the current version along with the ESP Studio from the previous version.

### 1.10 Input and Output Adapters

Input and output adapters enable Event Stream Processor to send and receive messages from dynamic and static external sources and destinations. Most adapters provided can be used as input or output adapters.

Input adapters connect to an external datasource and translate incoming messages from the external sources into a format that is accepted by the ESP server. Output adapters translate rows processed by Event Stream Processor into message formats that are compatible with external destinations and send those messages downstream. See the figure for an example.
For a complete list of adapters supplied by Event Stream Processor, see the SAP Event Stream Processor: Adapters Guide.

1.10.1 Custom Adapters

In addition to the adapters provided by Event Stream Processor, you can write your own adapters to integrate into the server.

SAP Event Stream Processor provides a variety of SDKs that allow you to write adapters in a number of programming languages, including:

- C
- C++
- Java
- .NET (C#)

For detailed information about creating custom adapters using the SDKs, see the SAP Event Stream Processor: Building Custom Adapters and SAP Event Stream Processor: SDK Guide.

You can also create custom adapters using the Event Stream Processor adapter toolkit. For detailed information, see the SAP Event Stream Processor: Building Custom Adapters.

1.11 Authoring Methods

SAP Event Stream Processor Studio provides visual and text authoring environments for developing projects.

In the visual authoring environment, you can develop projects using graphical tools to define streams and windows, connect them, integrate with input and output adapters, and create a project consisting of queries.
In the text authoring environment, you can develop projects in the Continuous Computation Language (CCL), as you would in any text editor. Create data streams and windows, develop queries, and organize them in hierarchical modules and projects.

You can easily switch between the Visual editor and the CCL editor at any time. Changes made in one editor are reflected in the other.

In addition to its visual and text authoring components, the Studio includes environments for working with sample projects, and for running and testing applications with a variety of 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 run in the Studio regardless of the operating system on which it was compiled.

You can access all ESP Studio components and features from within SAP HANA studio, as well as some SAP HANA-specific features, using the SAP Event Stream Processor plugin for SAP HANA studio.

If you prefer to work from the command line, you can develop and run projects using the `streamingproject`, `streamingprojectclient`, and `streamingcompiler` commands. For a full list of Event Stream Processor utilities, see the SAP Event Stream Processor: Utilities Guide.

### 1.12 Continuous Computation Language

CCL is the primary event processing language of SAP Event Stream Processor. Projects are defined in CCL.

CCL is based on Structured Query Language (SQL), 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.

The key distinguishing feature of CCL is its ability to continuously process dynamic data. A SQL query typically executes only once each time it is submitted to a database server and must be resubmitted every time a user or an application needs to reexecute the query. By contrast, a CCL query is continuous. Once it is defined in the project, it is registered for continuous execution and stays active indefinitely. When the project is running in the ESP 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 ESP server does not use a SQL query engine. Instead, it compiles CCL into a highly efficient byte code that is used by the ESP server to construct the continuous queries within the data-flow architecture.

CCL queries are converted to an executable form by the CCL compiler. ESP servers are optimized for incremental processing, hence the query optimization is different than for databases. Compilation is typically performed within Studio, but it can also be performed by invoking the CCL compiler from the command line.
1.13  CCLScript

CCLScript is a scripting language that brings extensibility to CCL, allowing you to create custom operators and functions that go beyond standard SQL.

The ability to embed CCLScript scripts in CCL provides tremendous flexibility, and the ability to do it within the CCL editor maximizes user productivity. CCLScript also allows you to define any complex computations that are easier to define using procedural logic rather than a relational paradigm.

CCLScript is a simple scripting language comprised 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 or Perl.

Related Information

Flex Operators [page 113]

*Flex operators are custom operators that let you write CCLScript scripts to operate on incoming events.*
2 Cluster Connectivity in Studio

In Studio, you can run a project on either a local or a remote cluster, using any of three methods of authentication. You can automatically run a project on a local cluster without needing to perform any prior steps. To run a project on a remote cluster, however, first connect to that cluster. This requires some simple configuration steps.

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.

A local cluster allows users to work on projects from their local machine. Use a local cluster to develop and test a project, but do not use a local cluster in production. Disconnecting a local cluster also stops the cluster. Internet access is not required.

A remote cluster allows users to connect to a server that is more powerful than the default server. The ability to use manual input, playback, and other Studio features is available. A remote cluster also allows users to share a project within the cluster with other users.

To run a project on a remote cluster, the remote cluster connection must first be configured in Studio. The administrator of the remote cluster must start it outside of the Studio. Once the cluster is running, you can connect to it from Studio and run the project.

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.

When you have Studio installed on a separate machine from the SAP ESP server, you cannot run projects on the default local cluster. Instead, connect to the SAP ESP server and then run your project. See Connecting to a Remote Cluster [page 20] for detailed instructions.

2.1 Changing Networking Preferences

Modify the default preferences for how the machine running ESP Studio connects with other ESP machines.

Context

SAP Event Stream Processor 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) you will have to modify the network preferences for Studio.

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.
4. Click Apply to save your new settings.
5. Click OK to exit.

2.2 Running a Project from a Local Cluster

Connect SAP Event Stream Processor Studio to the local cluster and run the project there.

Context

Run ESP Project enables you to run projects on a local or remote cluster from either the SAP ESP Authoring perspective or the SAP ESP Run-Test perspective.

When you have Studio installed on a separate machine from the SAP ESP server, you cannot run projects on the default local cluster. Instead, connect to the SAP ESP server and then run your project. See Connecting to a Remote Cluster [page 20] for detailed instructions.

Procedure

1. In the SAP ESP Authoring perspective.
   a. Select a project and open it in either the Visual Editor or the CCL Editor.
   b. Select Run ESP Project.

   i  Note

   If you are prompted for a user name and password for the local cluster, you must first change the default system-generated credentials. See Prompted for Local Cluster Password in Troubleshooting.

   The Server View in the SAP ESP Run-Test perspective opens, showing the project connection. A successful connection shows the server streams below the server folder, and the Console shows the server log for the project.

   If the connection is unsuccessful, you see a Server Connection error dialog.

2. In the SAP ESP Run-Test perspective.
   a. Select Run ESP Project.

   i  Note

   If you are prompted for a user name and password for the local cluster, you must first change the system-generated credentials. See Prompted for Local Cluster Password in Troubleshooting.

   The system displays a list of projects in the Select Project pop-up window.
b. Select the project that you want to run.

i Note

Select the drop-down menu to the immediate right of Run ESP Project to bring up the list of running or recently opened projects.

2.3 Configuring the Local Cluster for a Remote Connection

Edit the cluster configuration file for the Studio local cluster to allow connections from remote machines.

Procedure

1. Open the cluster configuration file, located at \texttt{STREAMING\_HOME/studio/clustercfg/studio.xml}.
2. Set the value of the \texttt{<STREAMING\_HOSTNAME>} macro to a hostname or IP address. For example:

\[
\text{<Macro type="value" name="STREAMING\_HOSTNAME">abc.sap.corp</Macro>}
\]

Results

You can now access projects started on the Studio cluster from other machines on the network by providing the cluster credentials.

2.4 Connecting to a Remote Cluster

Connect to a remote cluster from Studio to run a project on the cluster.

Prerequisites

The cluster’s administrator must have started the remote cluster outside of SAP Event Stream Processor Studio. If using Kerberos authentication, run a program outside of SAP Event Stream Processor Studio to obtain a current Ticket Granting Ticket (TGT).
Procedure

1. Select the SAP ESP 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

   i Note
   If this remote cluster employs user/password authentication, you will be prompted to provide the required user name and password. Studio does not store this information.

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.

2.5 Connecting to a Kerberos-Enabled Server

Connect to a remote server using Kerberos authentication.

Prerequisites

The system administrator must have provided the necessary elements for connecting to a Kerberos enabled server: Key Distribution Center, Kerberos Realm, Service, User name, and Cache.

Procedure

2. Fill the Key Distribution Center, Kerberos Realm, Service, User name, and Cache fields based on information provided by your system administrator.
3. Click Apply.
4. Click OK to exit Studio Preferences.
2.6 Connecting to an RSA-Enabled Server

Connect to a remote server using RSA authentication.

Prerequisites

The system administrator must have provided the necessary elements for connecting to an RSA enabled server: RSA User, Keystore Password and RSA Keystore.

Procedure

2. Enter the following information:
   - **RSA User**: Provide the user name of the keystore.
   - **Keystore Password**: Provide the password of the keystore.
   - **RSA Keystore**: Provide the name of the keystore file.
3. Click **Apply**.
4. Click **Ok** to exit Studio Preferences.
5. Enter the following command to import the keystore to the PKCS12 type store:
   ```
   $JAVA_HOME/bin/keytool -importkeystore -srckeystore keystore.jks -destkeystore keystore.p12 -deststoretype PKCS12
   ```
   Creates a PKCS12 keystore.
6. Enter the following command to extract a pem format private key:
   ```
   openssl pkcs12 -in keystore.p12 -out keystore.private -nodes
   ```
   Creates a private key.
7. Copy the private key file to the directory where the keystore file is located.
8. In the Server View, connect to a remote cluster using RSA authentication.
2.7 Configuring a Remote Cluster Connection

Use SAP Event Stream Processor Studio preferences to manage remote cluster connections and authentication methods.

Prerequisites

The administrator of the remote cluster must have provided the necessary information about the cluster: host name, port number, authentication method, and, if using RSA, the RSA user, password and keystore.

Procedure

1. To add a new remote cluster connection, select New Server URL in the Server View toolbar.

   ![Note]
   
   In the Server View toolbar, you can also select Studio Preferences and add a new connection through SAP Event Stream Processor Run Test and click Run Test. Select New.

   Studio displays the New Server screen.
2. In the fields provided, enter the host name and port number for the cluster connection:

   Host: myserver.mycompany.com
   Port: 12345
3. (Optional) To enable encryption for Cluster Manager connections, select SSL.
4. Select an authentication method: Kerberos, RSA, or User/Password.
5. If you selected RSA, enter the following information:

   - **RSA User**: Provide the key alias.
   - **RSA Password**: Provide the keystore password.
   - **RSA Key store**: Provide the file name for the key store which contains the private key.
6. Click OK.

Results

In the SAP ESP Run-Test perspective, the Server view accesses the list of stored server connections. Depending on the authentication method, Studio attempts to connect immediately (for RSA and Kerberos modes), or shows a login dialog for each cluster configured for User/Password authentication.
2.8 Modifying a Remote Cluster Connection

Change the authentication settings of a remote cluster connection that is already configured.

Context

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

Procedure

1. In the Server View, select Studio Preferences > SAP Event Stream Processor Studio > Run Test. Studio displays the Run Test screen.
2. Select an existing server connection. The Remove and Edit buttons are activated.
4. Make your changes and click OK. Studio displays the Run Test screen.
5. Click OK to save your changes.
2.9 Managing Secure Storage Credentials for an ESP Server Connection

Use secure storage to encrypt user name and password credentials for an SAP Event Stream Processor server connection in Studio. Secure storage enables credential information for the SAP ESP server connections to persist in Studio.

2.9.1 Storing Credentials in Secure Storage for an ESP Server Connection

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

Context

You are prompted to enter user name and password credentials in three situations: the first time you connect to a server in Studio; if you choose not to persist credential information; or 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 Event Stream Processor Credentials in Secure Storage. It is recommended that you click the checkbox to store the user name and password information you entered. If you do not choose to store the credentials using the Store Event Stream Processor 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 Event Stream Processor > Run Test.
3. Click on the checkbox for Use Secure Storage for ESP Credentials.
4. Click Apply then OK.
   The user name and password credentials you entered when prompted are saved in \C:\Users\uname\eclipse\org.eclipse.equinox.security\secure_storage.
2.9.2 Modifying Credentials in Secure Storage for an ESP Server Connection

Modify user name and password credentials in secure storage for an SAP Event Stream Processor server connection. Change user name and password information as required.

Context

Modify credential information in secure storage:

Procedure

1. In the main menu, select Windows > Preferences.
2. Open the Secure Storage preference page. Either:
   ○ Select SAP Event Stream Processor > Run Test and clicking on the Secure Storage Settings link, or,
   ○ Select General > Security > 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.
3  Getting Started with a Project

To begin developing a project, start the SAP Event Stream Processor Studio, review Studio basics, and optionally step through an example before creating your own project.

3.1  Launching Studio

Start SAP Event Stream Processor Studio using the desktop shortcut, Windows Start menu, or the command line. If you installed the ESP plugin for SAP HANA studio, you can access SAP Event Stream Processor functionality within the SAP HANA studio by opening the SAP ESP Run-Test and SAP ESP Authoring perspectives.

Procedure

1. In SAP HANA studio, go to Window ➤ Open Perspective ➤ Other, then open the SAP ESP Authoring or SAP ESP Run-Test perspective.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>In SAP HANA studio, go to Window ➤ Open Perspective ➤ Other, then open the SAP ESP Authoring perspective or the SAP ESP Run-Test perspective. From outside of SAP HANA studio, either: ○ Double-click the SAP ESP Studio shortcut on your computer desktop, or, ○ Select Event Stream Processor Studio from the Start menu.</td>
</tr>
<tr>
<td>Linux or UNIX</td>
<td>In SAP HANA studio, go to Window ➤ Open Perspective ➤ Other, then open the SAP ESP Authoring perspective or the SAP ESP Run-Test perspective. From outside of SAP HANA studio, either: ○ Double-click the SAP ESP Studio shortcut on your computer desktop, or, ○ At the command line, enter $STREAMING_HOME/studio/streamingstudio.</td>
</tr>
</tbody>
</table>

3.2  Studio Basics

In the Studio, you use different perspectives and views to run examples, create and edit projects, and run and test your projects in a running Event Stream Processor server.

The Studio is based on the Eclipse IDE and uses some Eclipse terms:
A named window in the Studio user interface. Views are tabbed so several can overlap, occupying the same pane. The SAP ESP 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.

A named set of views. Like views, perspectives are tabbed so several can use the same space. Perspectives in Studio include SAP ESP Authoring and SAP ESP Run-Test.

The Studio workspace is a directory that stores projects. The Studio workspace is distinct from cluster workspaces, which are logical groupings of deployed projects.

By default, all perspectives are open. To switch to another perspective, click its tab, just below the main menu bar.

### Table 2: User Activities in Studio Perspectives

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP ESP Authoring</td>
<td>• Create and edit projects&lt;br&gt;• Develop projects and diagrams in the Visual editor, a graphical editing environment&lt;br&gt;• Develop projects in the CCL editor, a text-oriented editing environment where you edit CCL code&lt;br&gt;• Compile projects</td>
</tr>
<tr>
<td>SAP ESP Run-Test</td>
<td>• Start and connect to servers&lt;br&gt;• Run projects&lt;br&gt;• Enter test data by uploading data files to a server, or entering data manually to a stream&lt;br&gt;• Publish data&lt;br&gt;• Execute a query against a running project&lt;br&gt;• Use the Event Tracer and Debugger to set breakpoints and watchpoints, and trace the flow of data through a project&lt;br&gt;• Record incoming event data to a playback file, and play back captured data into a running project&lt;br&gt;• Monitor performance</td>
</tr>
</tbody>
</table>

### Related Information

Visual Editor Authoring [page 46]

*The Visual editor lets you create and edit projects without learning CCL syntax.*
Running and Testing Projects [page 147]

Test a project by compiling and running it on a server, accessing and filtering streams, saving and uploading data to the SAP Event Stream Processor server, and setting project configurations.

3.2.1 Project Explorer

Organize and navigate among your projects using the Project Explorer, which provides a tree-structured hierarchy of folders and files.

The Project Explorer view lets you organize project files, navigate to files and perform various file-based actions:

- Create new CCL files
- Create new projects
- Edit existing files
- Delete files
- Create new folders

The Toggle ESP Nature option in the context menu toggles the appearance of ESP-specific context menu options for individual projects in the Project Explorer. When using ESP in the SAP HANA studio environment, use this option to enable or disable ESP options as necessary.

When you add a new ESP project, it displays 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.

Related Information

Editing a Project in the Visual Editor [page 52]

In the visual editor, a diagram represents the elements of a project. Add or remove shapes and connections or edit shape attributes to make changes to a project.

Editing in the CCL Editor [page 134]

Update and edit CCL code as text.

3.3 The Studio Log File

SAP Event Stream Processor 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.

### 3.4 Creating a Project

Use the SAP Event Stream Processor Studio to create new projects that can run on the ESP Server.

#### Context

Continuous queries are organized into projects that also define inputs, outputs, a schema and other options for processing event data.

#### Procedure

1. Select File > New > SAP ESP Project...
2. Enter a valid project name:
   - Must start with a lowercase letter, underscore, or dollar sign.
   - All other characters must be lowercase letters, numbers, underscores, or dollar signs.
   - Must not contain spaces.
   
   For example, enter myfirstproject.
3. In the Directory field, accept the default location or browse to a directory in which to store the new project folder.

   ESP 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," ESP 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.
Related Information

Opening a Project in the Workspace [page 31]
Open a project from Project Explorer when it already exists in your workspace.

Importing an Existing Project [page 32]
Import an existing Event Stream Processor project from another location into your workspace.

Editing a Project in the Visual Editor [page 52]
In the visual editor, a diagram represents the elements of a project. Add or remove shapes and connections or edit shape attributes to make changes to a project.

Switching Between the CCL and Visual Editors [page 56]
Change between the two editors to maximize Studio’s flexibility for creating and editing a project.

3.4.1 Opening a Project in the Workspace

Open a project from Project Explorer when it already exists in your workspace.

Procedure

1. In Project Explorer, expand project folders to see project files.
2. Double-click a file to open it for editing.
   ○ .ccl files open in the CCL editor
   ○ .cclnotation files open in the Visual editor

   **Note**
   If you receive 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.

   You cannot have both the .cclnotation and .ccl files for the same project open at the same time.

Related Information

Creating a Project [page 30]
Use the SAP Event Stream Processor Studio to create new projects that can run on the ESP Server.

Importing an Existing Project [page 32]
Import an existing Event Stream Processor project from another location into your workspace.

Editing a Project in the Visual Editor [page 52]
In the visual editor, a diagram represents the elements of a project. Add or remove shapes and connections or edit shape attributes to make changes to a project.
Switching Between the CCL and Visual Editors [page 56]
Change between the two editors to maximize Studio’s flexibility for creating and editing a project.

3.4.2 Importing an Existing Project

Import an existing Event Stream Processor project from another location into your workspace.

Context

This procedure does not apply to the ESP Plugin for SAP HANA Studio. To import projects into SAP HANA, see Importing Multiple Projects [page 33].

Procedure

1. Choose File → Open → ESP Project.
2. Browse to the root directory of the project.
3. (Optional) Select Copy projects into workspace.
   ○ Copy projects into workspace copies the project in the workspace and opens it from there. Changes are made to the copy only.
   ○ If this option is not checked, the project opens in its original location.
4. Click Finish.

Related Information

Creating a Project [page 30]
Use the SAP Event Stream Processor Studio to create new projects that can run on the ESP Server.

Opening a Project in the Workspace [page 31]
Open a project from Project Explorer when it already exists in your workspace.

Editing a Project in the Visual Editor [page 52]
In the visual editor, a diagram represents the elements of a project. Add or remove shapes and connections or edit shape attributes to make changes to a project.

Switching Between the CCL and Visual Editors [page 56]
Change between the two editors to maximize Studio’s flexibility for creating and editing a project.
3.4.3 Importing Multiple Projects

If you have multiple projects existing in the same directory outside of your default workspace, you can import all of those projects to your workspace at once.

Context

When importing projects, you can copy them into your workspace, or point to their original location. If you make copies, changes you make to the workspace copies are not reflected in the original location.

Procedure

1. In the SAP ESP Authoring perspective, right-click the Project Explorer and select Import from the context menu.
2. In the Import dialog, expand the General folder and click Existing Projects into Workspace.
3. Click Next.
4. Enable the Select root directory option and enter or browse to the root directory containing the projects you want to import.
5. (Optional) Clear the check mark from any projects you do not want to import.
6. (Optional) Clear the Copy projects into workspace option.
7. Click Finish.

3.5 Renaming a Project

Change the project name for easier recognition, or to reflect any changes in the content.

Prerequisites

You must be in the SAP ESP Authoring perspective.

Context

Follow these steps to rename a project that you have created in Studio.
Procedure

1. Select the project folder from the Project Explorer field.
2. Right click and select Rename.
   The same naming rules that apply when creating a project also apply when renaming a project:
   ○ Must start with a lowercase letter, underscore, or dollar sign.
   ○ All other characters must be lowercase letters, numbers, underscores, or dollar signs.
   ○ Must not 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.

3.6 Deleting a Project

Delete unused projects from the Studio.

Prerequisites

You must be in the SAP ESP Authoring perspective.

Procedure

1. Navigate to 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 ESP workspace and from your machine.
   If this option is not checked, ESP deletes the project from Project Explorer, but does not remove the project from your workspace directory.
5. Click OK.
3.7  Event Stream Processing with SAP HANA

SAP Event Stream Processor integrates with SAP HANA in a number of ways, providing multiple methods for sharing data, all of which you can tailor to fit your projects.

3.7.1  Creating a Project in SAP HANA Studio

Save SAP ESP projects in the SAP HANA studio workspace directory.

Context

You can also open or import multiple existing projects from an ESP Studio workspace to an SAP HANA studio workspace. See Opening a Project in SAP HANA Studio [page 36] for more information.

Procedure

1. In SAP HANA studio menu bar, go to File ➤ New ➤ Project..., then select SAP Event Stream Processor ➤ New ESP Project.
2. In the Name field, enter a project name. A valid project name:
   ○ Must start with a lowercase letter, underscore, or dollar sign.
   ○ All other characters must be lowercase letters, numbers, underscores, or dollar signs.
   ○ Must not contain spaces.
3. In the Directory field, accept the default location or browse to a directory in which to store the new project folder. In the ESP Studio plugin, the default directory is the SAP HANA studio workspace 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.
3.7.2 Opening a Project in SAP HANA Studio

Open an existing project in the SAP HANA studio.

Procedure

1. In the SAP HANA studio menu bar, click > Windows > Open Perspective > Other > SAP ESP Authoring.
2. Click OK.
3. In the Studio toolbar, click the Open ESP Project icon.
4. Browse to the root directory of the project.
5. (Optional) Select Copy projects into workspace.
   - Copy projects into workspace copies the project in the workspace and opens it from there. Changes are made to the copy only.
   - If this option is not checked, the project opens in its original location.
6. Click Finish.

3.7.3 Importing a Project into SAP HANA Studio

Import an existing project into the SAP HANA workspace.

Procedure

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

When you install the ESP plugin for SAP HANA studio, you can 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 SAP ESP project.

The new reference can then be used in a join along with SAP Event Stream Processor 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.

3.7.5 Reference Queries for HANA Tables and Views

Reference table queries provide a way to augment the streaming data in a project with information from a table or view in SAP HANA.

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 external database, you can use reference table queries to look it up. There are two parts to this: creating the reference to an external table and using the reference to execute an ad hoc query in a join or flex operator.

When creating the reference, you must first decide what data you want to use. Then identify the external database 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, you need to decide what you want as the output of the join. There are numerous restrictions on how you can 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 must specify the primary key of the reference and use the complete primary key in the \texttt{ON} or \texttt{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 utilize a reference in the same ways you can utilize 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.

To learn more about creating and configuring reference table queries, see Reference Table Queries [page 89].
3.7.6 Creating a Database Table

Using SAP HANA studio, create a table in the SAP HANA database.

Procedure

1. From the SAP HANA studio, open the SAP ESP Authoring perspective.
2. Open, or create, the project from which you want to create your SAP HANA table.
3. Select the ESP project element from which to create the SAP HANA table. This feature is supported on all input and output elements except Delta Streams, Error Streams, and Reference Table Queries. Local elements do not support this feature.
4. Display the context menu and select the Create Database Table option.
5. In the Service field, accept the default value, or click the Discover button and select a value from the list. Only services for which the Enable as Hana Reference Service property was checked when they were defined are displayed in the list.
6. In the Target Schema field, click the Discover button and select a value from the list.
7. In the Target Table field, accept the default value, type in a new value, or click the Discover button and select a value from the list.
8. In the Table Modification section, select New Table or Replace Table. An error will be generated if you choose New Table and the table already exists in the selected schema, or if the table does not exist in the selected schema and you have chosen Replace Table.
9. In the Table Kind section, select Row or Column.
10. Click OK.

3.7.6.1 Datatype Mapping to SAP HANA Tables

Reference information on how SAP Event Stream Processor datatypes map to SAP HANA datatypes.

For each datatype in the ESP stream or window from which you are creating an SAP HANA table, this table shows what datatype will be created in the SAP HANA table.

<table>
<thead>
<tr>
<th>Event Stream Processor Datatypes</th>
<th>SAP HANA Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigdatetime</td>
<td>timestamp</td>
</tr>
<tr>
<td>binary</td>
<td>varbinary(5000)</td>
</tr>
<tr>
<td>boolean</td>
<td>tinyint</td>
</tr>
<tr>
<td>seconddate</td>
<td>date</td>
</tr>
<tr>
<td>double</td>
<td>real</td>
</tr>
<tr>
<td>fixed_decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>int32</td>
<td>integer</td>
</tr>
<tr>
<td>int64</td>
<td>bigint</td>
</tr>
<tr>
<td>Event Stream Processor Datatypes</td>
<td>SAP HANA Datatypes</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>interval</td>
<td>bigint</td>
</tr>
<tr>
<td>money</td>
<td>decimal(19,4)</td>
</tr>
<tr>
<td>moneyX</td>
<td>decimal(19,X)</td>
</tr>
<tr>
<td>string</td>
<td>nvarchar(5000)</td>
</tr>
<tr>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td>msdate</td>
<td>timestamp</td>
</tr>
</tbody>
</table>
4   Exposing a Project as a Web Service Using the Web Services Provider

Set the Web Service Enabled option to true in the project configuration (ccr) file to expose a project as a Web service using one of the SAP Event Stream Processor Web Services providers.

Prerequisites

- (For the SOAP Web Services Provider) Start the SOAP server for the SAP Event Stream Processor Web Services Provider.
- To run a project in a workspace other than the default, ensure that one or more connected workspaces are available.

Procedure

1. Select File > Open > Project... and open the project you want to run.
2. Double-click the project configuration file (<projectname>.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 options are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-HA</td>
<td>Non-HA deployments create one project option item and one instance item as children under the project deployment item.</td>
</tr>
<tr>
<td>HA</td>
<td>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.</td>
</tr>
</tbody>
</table>

5. Set the value of the Web Service Enabled option to true in the Project Deployment Details window.
7. For the changes to take effect, use Studio or streamingclusteradmin to stop and remove the project from the node, then redeploy (add) the project. Or restart the cluster on which the project runs.
8. To run the project, either:
   - Click Run ESP Project in the main toolbar (in either the SAP ESP Authoring or the SAP ESP Run-Test perspective) to run the project in the default workspace, or,
   - Click the drop-down arrow next to the Run Project tool and choose Run ESP Project in Workspace. Then select the workspace where this project will run.

The project runs and shows results in the SAP ESP Run-Test perspective.
5 Adapter Support for Schema Discovery

Lists all adapters currently available from SAP, whether they support schema discovery, and if so, the properties they use to enable it.

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

<table>
<thead>
<tr>
<th>Adapter</th>
<th>Supports Schema Discovery</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP ASE Output</td>
<td>Yes</td>
<td>DB Service Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name of the database service that represents the SAP ASE database into</td>
</tr>
<tr>
<td></td>
<td></td>
<td>which information will be loaded.</td>
</tr>
<tr>
<td>Atom Feed Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Database Input</td>
<td>Yes</td>
<td>Database Service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of database service from which the adapter obtains the database connection.</td>
</tr>
<tr>
<td>Database Output</td>
<td>Yes</td>
<td>Database Service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of service entry to use.</td>
</tr>
<tr>
<td>SAP Event Stream Processor</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Add-In for Microsoft Excel</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>SAP Event Stream Processor</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Web Services Provider</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>File/Hadoop CSV Input</td>
<td>Yes</td>
<td>File Input Transporter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Dir</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● AccessMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● (Optional) ScanDepth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSV String to ESP Formatter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ExpectStreamNameOpcode</td>
</tr>
<tr>
<td>File/Hadoop CSV Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>File FIX Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>File FIX Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>File/Hadoop JSON Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>File/Hadoop JSON Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>File/Hadoop XML Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>File/Hadoop XML Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Adapter</td>
<td>Supports Schema Discovery</td>
<td>Properties</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>File/Hadoop XML Event Input</td>
<td>Yes</td>
<td>File Input Transporter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Dir</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AccessMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• (Optional) ScanDepth</td>
</tr>
<tr>
<td>File/Hadoop XML Event Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>FIX Session Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Flex Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>FTP CSV Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>FTP CSV Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>FTP Event XML Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>FTP Event XML Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>HTTP Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>HTTP Client Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>JDBC Input</td>
<td>Yes</td>
<td>JDBC Input Transporter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Host</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DbName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DbType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DbDriver</td>
</tr>
<tr>
<td>JDBC Output</td>
<td>—</td>
<td>JDBC Output Transporter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Host</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DbName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DbType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DbDriver</td>
</tr>
<tr>
<td>Adapter</td>
<td>Supports Schema Discovery</td>
<td>Properties</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>JMS CSV Input</td>
<td>Yes</td>
<td>JMS Input Transporter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ConnectionFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JndiContextFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JndiURL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DestinationType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DestinationName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MessageType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• (Optional) ScanDepth</td>
</tr>
<tr>
<td>JMS CSV Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>JMS FIX Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>JMS FIX Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>JMS Object Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>JMS Object Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>JMS Event XML Input</td>
<td>Yes</td>
<td>JMS Input Transporter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ConnectionFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JndiContextFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JndiURL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DestinationType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DestinationName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MessageType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• (Optional) ScanDepth</td>
</tr>
<tr>
<td>JMS Event XML Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>KDB Input</td>
<td>Yes</td>
<td>• KDB Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KDB Port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KDB User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KDB Password</td>
</tr>
<tr>
<td>KDB Output</td>
<td>Yes</td>
<td>• KDB Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KDB Port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KDB User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KDB Password</td>
</tr>
<tr>
<td>Log File Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>NYSE Technologies Input</td>
<td>Yes</td>
<td>Discovery Directory Path</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute path to the adapter discovery directory.</td>
</tr>
<tr>
<td>Open Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Open Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Adapter</td>
<td>Supports Schema Discovery</td>
<td>Properties</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>Random Tuples Generator Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Replication Server Input</td>
<td>Yes</td>
<td>• RSSD Host</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RSSD Port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RSSD Database Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RSSD User Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RSSD Password</td>
</tr>
<tr>
<td>RTView Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>SAP HANA Output</td>
<td>Yes</td>
<td>DB Service Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name of the database service that represents the SAP HANA database into which information will be loaded.</td>
</tr>
<tr>
<td>SAP RFC Input</td>
<td>Yes</td>
<td>• Adapter Configuration File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adapter Mapping File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP Host</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP System Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP Client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Username</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td>SAP RFC Output</td>
<td>Yes</td>
<td>• Adapter Configuration File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adapter Mapping File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP Host</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP System Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP Client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Username</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td>SAP IQ Output</td>
<td>Yes</td>
<td>DB Service Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name of the database service that represents the IQ database into which information will be loaded.</td>
</tr>
<tr>
<td>SMTP Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket FIX Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket FIX Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket CSV Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket CSV Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket JSON Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket JSON Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Adapter</td>
<td>Supports Schema Discovery</td>
<td>Properties</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Socket Event XML Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Socket Event XML Output</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Tibco Rendezvous Input</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>Tibco Rendezvous Output</td>
<td>No</td>
<td>—</td>
</tr>
</tbody>
</table>
| Web Services (SOAP) Input Adapter | Yes | ● Adapter Configuration File  
● Adapter Mapping File  
● Discovery WSDL URL  
● Discovery Working Directory  
● Discovery Service Name |
| Web Services (SOAP) Output Adapter | Yes | ● Adapter Configuration File  
● Adapter Mapping File  
● Discovery WSDL URL  
● Discovery Working Directory  
● Discovery Service Name |
| WebSphere MQ Input | No | — |
| WebSphere MQ Output | No | — |

**Note**

The adapter does not support schema discovery if you are using HTTP Basic Access Authentication.

### Related Information

**Schema Discovery** [page 59]

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

**Discovering a Schema** [page 60]

*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.*
6 Visual Editor Authoring

The Visual editor lets you create and edit projects without learning CCL syntax.

It is also a valuable tool for experienced CCL programmers, particularly when working on complex projects, as a way to easily visualize the data flow and navigate within the project. In the Visual editor, the project is represented by one or more diagrams that show streams, windows, adapters, and the data flows between them.

Begin by developing a simple project. Use the graphical tools to add streams and windows, connect them, and associate them with adapters. Add simple queries directly in the diagram using the visual editing tools.

Once you have a basic diagram completed, compile and run your project.

When you are confident that your simple project is working, you can progress to advanced features: more complex queries, Flex operators for custom operations, modularity, and custom adapters. You can access many of these features in the visual authoring environment.

For more complex queries and other advanced features, you can switch to the CCL editor. 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.

6.1 Diagrams

In visual authoring, you use diagrams to create and manipulate the streams, windows, connections, and other components of a project, and create simple queries.

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.

**Related Information**

*Shape Reference* [page 49]
*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.*

*Changing the Display of Diagrams* [page 56]
*Display diagrams in verbose or iconic mode. Lay out the elements in the diagram left to right or top down.*

**6.2 The Authoring Perspective**

The visual editor, CCL editor, and other tools and views in the SAP ESP Authoring perspective allow you to create, view, and edit a diagram or CCL file.

When viewing the SAP ESP Authoring perspective, its components are:

- **Editor (1)** (Center) Authoring 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.

- **Palette (2)** (Right side of Editor) Includes groups of tools used to create new CCL elements on the diagram. Most shapes on the Palette correspond to a CCL statement.
Project Explorer (3) provides a hierarchical tree structure of folders and files.

Properties view (4) displays the properties of the object selected in the diagram. You can also set properties in this view, and edit expressions.

Outline view (5) 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 (6) 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 (7) provides full-text search capability for finding text strings in the workspace. Useful in navigating File Explorer, and project contents in the CCL editor. You can filter search results, and copy, remove, or replace results found.

Problems (8) displays errors found when you compile a project.

Console (9) displays messages generated when interacting with ESP components.

Figure 4: SAP ESP Authoring Perspective Views

You can customize the arrangement of views in your perspectives. See Customizing the Studio Work Environment in the SAP Event Stream Processor: Studio Users Guide.
6.3 Shape Reference

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.

Table 3: Shapes in the Palette

<table>
<thead>
<tr>
<th>Shape</th>
<th>Purpose</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Connector]</td>
<td>Creates flows between streams and windows, establishes references between streams and shared components, or attaches notes to shapes.</td>
<td>Click to select the connector tool, then click each of the shapes in the diagram to be connected.</td>
</tr>
<tr>
<td>![Note]</td>
<td>Creates a comment on the diagram only. This comment does not appear in the CCL file.</td>
<td>Documents additional user-generated information in the SAP ESP Authoring perspective for a project in Studio.</td>
</tr>
<tr>
<td>![Input Stream]</td>
<td>The entry point for unkeyed event streams into a project. Receives data from either an input adapter or an external publisher.</td>
<td>A stream does not retain any data and does not have a state. Data in an input stream is not keyed.</td>
</tr>
<tr>
<td>![Derived Stream (Local)]</td>
<td>Applies a continuous query to data arriving from another stream or window to produce a new stream.</td>
<td>Streams do not retain data and do not have keys. They are &quot;insert only,&quot; 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.</td>
</tr>
<tr>
<td>![Derived Stream (Output)]</td>
<td>Applies a continuous query to data arriving from another stream or window. Retains data, and retention rules can be set.</td>
<td>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.</td>
</tr>
<tr>
<td>Shape</td>
<td>Purpose</td>
<td>Usage</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Input Window</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Flex</td>
<td>A programmable operator that uses custom CCLScript scripts to process incoming events.</td>
<td>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).</td>
</tr>
<tr>
<td>Aggregate</td>
<td>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.</td>
<td>Always creates a new window. Requires a GROUP BY element. You can optionally set window size using retention rules.</td>
</tr>
<tr>
<td>Compute</td>
<td>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.</td>
<td>Produces a derived stream when the input is a stream. Produces a derived window when the input is a delta stream or window.</td>
</tr>
<tr>
<td>Filter</td>
<td>Takes input from a single source and applies a filter. Creates a stream of records that match the filter criteria.</td>
<td>Produces a derived stream when the input is a stream. Produces a derived window when the input is a delta stream or window.</td>
</tr>
<tr>
<td>Join</td>
<td>Takes input from two or more sources and joins them based on common data elements.</td>
<td>See related information in this guide and the SAP Event Stream Processor: Developer Guide for join support details.</td>
</tr>
<tr>
<td>Pattern</td>
<td>Takes input from two or more sources and detects patterns of events. One output record is produced every time a pattern is detected.</td>
<td>Detects patterns in the incoming data. See related information in the SAP Event Stream Processor: Developer Guide.</td>
</tr>
<tr>
<td>Union</td>
<td>Merges input from two or more sources. One output record is produced for every input record.</td>
<td>All inputs must have a common schema.</td>
</tr>
<tr>
<td>Shape</td>
<td>Purpose</td>
<td>Usage</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Named Schema</td>
<td>Represents a CCL CREATE SCHEMA statement. Reusable definition of column structure that can be referenced by streams and windows.</td>
<td>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.</td>
</tr>
<tr>
<td>Module</td>
<td>Represents a CCL CREATE MODULE statement. Creates a new module that can be used in one or more places in the project.</td>
<td>A module can contain all the same elements as a project and provides for reuse.</td>
</tr>
<tr>
<td>Log Store</td>
<td>Stores data held in windows. Provides disk-based recovery but is slower than a memory store</td>
<td>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.</td>
</tr>
<tr>
<td>Memory Store</td>
<td>Stores data held in windows.</td>
<td>Faster than a log store but does not recover data after shutdown.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- (Default) Created implicitly by the CCL compiler, if no other store is specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- (Optional) Created explicitly, with windows assigned to specific stores, to optimize performance.</td>
</tr>
<tr>
<td>Input Adapters</td>
<td>Connects an input stream or input window to an external data source.</td>
<td>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.</td>
</tr>
<tr>
<td>Output Adapters</td>
<td>Connects an output stream or window to a destination.</td>
<td>Must be connected to either an output stream or an output window.</td>
</tr>
</tbody>
</table>
### Shape

<table>
<thead>
<tr>
<th>Shape</th>
<th>Purpose</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Reference" /> Reference</td>
<td>A CCL element that establishes a reference from an external database table to a project in ESP. Use references - in joins and in CCLScript programs - inside a module as well as within the main body of your project.</td>
<td>Can be used for data lookup and to enrich data streaming in SAP Event Stream Processor 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.</td>
</tr>
</tbody>
</table>

### Related Information

- **Simple Queries** [page 69]
  
  Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

- **Adding Shapes to a Diagram** [page 54]
  
  Create streams, windows, and shared components, relate them using continuous queries, and attach them to adapters.

- **Connecting Elements** [page 82]
  
  Connect two shapes in a diagram to create a data flow or link between them.

- **Join Types and Restrictions** [page 77]
  
  Determine what combination of attributes your join simple query must contain.

### 6.4 Editing a Project in the Visual Editor

In the visual editor, a diagram represents the elements of a project. Add or remove shapes and connections or edit shape attributes to make changes to a project.

**Procedure**

1. In the SAP ESP Authoring perspective, navigate to Project Explorer.
2. To open a saved project in the Visual editor, double-click the .cclnotation file name.
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. Save as you go (Ctrl+S).
   This saves changes to both the .cclnotation file (the diagram) and the .ccl file (the CCL).

5. To toggle between the Visual editor and the CCL editor, choose Switch to Text or Switch to Visual (F4).

6. To close the diagram, press Ctrl+W or Ctrl+F4, or click the X on the tab at the top of the editor.

---

**Related Information**

- **Creating a Project** [page 30]
  Use the SAP Event Stream Processor Studio to create new projects that can run on the ESP Server.

- **Opening a Project in the Workspace** [page 31]
  Open a project from Project Explorer when it already exists in your workspace.

- **Importing an Existing Project** [page 32]
  Import an existing Event Stream Processor project from another location into your workspace.

- **Switching Between the CCL and Visual Editors** [page 56]
  Change between the two editors to maximize Studio’s flexibility for creating and editing a project.

- **Project Explorer** [page 29]
  Organize and navigate among your projects using the Project Explorer, which provides a tree-structured hierarchy of folders and files.
6.5 Adding Shapes to a Diagram

Create streams, windows, and shared components, relate them using continuous queries, and attach them to adapters.

Procedure

1. Open a diagram in the Visual editor.
2. Click a shape tool in the Palette (*Input Window*, *Flex*, and so on), then click an empty area in the diagram. This creates the new shape in the diagram. Red borders indicate that the shape definition is incomplete or incorrect. When a shape definition is complete, the border changes to gray.
3. To view actions needed to complete a shape definition, hover the mouse over the shape in the diagram.

Next Steps

See tasks for specific shapes for more steps you may need to do.

Related Information

**Simple Queries** [page 69]
Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

**Shape Reference** [page 49]
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.

**Deleting an Element** [page 87]
Delete an element from the project to remove it completely, or delete it from the diagram only.

**Keyboard Shortcuts in the Visual Editor** [page 55]
Use keyboard shortcuts to access various functions quickly within the Visual editor.
6.6 Adding Comments to Shapes

Add comments to shapes in the Visual editor that will appear within a tooltip when you hover over them.

Prerequisites

‘Show comments in tooltip’ must be enabled in Preferences.

Procedure

1. In the visual editor, select a shape you want to add a comment for by clicking on it.
2. Once the shape is highlighted, select the comment field in the Properties view.
3. Click the ellipsis button and enter a comment into the box. Click OK when finished.

6.7 Keyboard Shortcuts in the Visual Editor

Use keyboard shortcuts to access various functions quickly within the Visual editor.

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

To add a keyboard shortcut:

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

Related Information

Adding Shapes to a Diagram [page 54]
Create streams, windows, and shared components, relate them using continuous queries, and attach them to adapters.

Deleting an Element [page 87]
Delete an element from the project to remove it completely, or delete it from the diagram only.
6.8 Changing the Display of Diagrams

Display diagrams in verbose or iconic mode. Lay out the elements in the diagram left to right or top down.

Prerequisites

Open the diagram in the Visual editor.

Procedure

- To toggle a shape between iconic and verbose mode:
  - In verbose mode, click the “minus” sign in the upper-left corner to collapse it.
  - In iconic mode, click the “plus” sign to expand it.
- To show all shapes as iconic or verbose, in the Visual editor toolbar click All Verbose or All Iconic.
- To change the orientation, in the Visual editor toolbar click Layout left to right or Layout top down.

Note

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

Related Information

Editing Studio Preferences [page 171]

Edit preferences to customize the SAP Event Stream Processor environment.

6.9 Switching Between the CCL and Visual Editors

Change between the two editors to maximize Studio’s flexibility for creating and editing a project.

Procedure

- To switch from the CCL editor to the Visual editor, right-click and choose Switch to Visual (F6), or click in the main toolbar.
To switch from the Visual editor to the CCL editor, right-click in the diagram and choose Switch To Text (F6), or click 📐 in the main toolbar.

Related Information

Creating a Project [page 30]
Use the SAP Event Stream Processor Studio to create new projects that can run on the ESP Server.

Opening a Project in the Workspace [page 31]
Open a project from Project Explorer when it already exists in your workspace.

Importing an Existing Project [page 32]
Import an existing Event Stream Processor project from another location into your workspace.

Editing a Project in the Visual Editor [page 52]
In the visual editor, a diagram represents the elements of a project. Add or remove shapes and connections or edit shape attributes to make changes to a project.

6.10 Building a Simple Project

Build a simple project entirely in the ESP Studio Visual editor by following the steps in linked tasks.

Prerequisites

Create a new project. See Creating a Project [page 30] or Creating a Project in SAP HANA Studio [page 35].

Context

Some tasks are optional. The order of tasks is approximate; each project differs in detail.

Tip
Work left to right, or top to bottom, starting with the inputs and then following the data flow. This strategy allows you to copy columns and column expressions into a new query from the input streams.

1. Adding an Adapter to a Project [page 58]
   Attach an adapter by inserting it in the diagram, connecting it to a stream or window, and setting properties.

2. Discovering a Schema [page 60]
   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.
3. **Adding an Input Stream or Window to a Project** [page 65]
   Input streams and windows accept data from a source external to the project.

4. **Connecting Elements** [page 82]
   Connect two shapes in a diagram to create a data flow or link between them.

5. **Setting Key Columns** [page 83]
   Set primary keys in the Visual editor within the Column compartment of the window, and Flex operator shapes.

6. **Editing Column Expressions** [page 84]
   Modify column expressions for windows and streams using an inline editor or dialog-based expression editor.

### Related Information

**Deleting an Element** [page 87]
Delete an element from the project to remove it completely, or delete it from the diagram only.

**Creating a Project** [page 30]
Use the SAP Event Stream Processor Studio to create new projects that can run on the ESP Server.

### 6.10.1 Adding an Adapter to a Project

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

#### Procedure

1. Open the **Input Adapters** or **Output Adapters** compartment in the Palette and use the up and down arrows to scroll through the list of adapters.
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:
     - 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. Either:
   - Select **Consolidate Adapter Properties from the adapter properties dialog**. When this setting is enabled, the system inspects the property set in the CCR for matching property pairings, then removes such
pairings 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. Or,

- Select **Use named property set** to use a named property set from the project configuration file, and then configure any properties that are not included in the property set, or,
- In the table, manually modify adapter properties.

### 6.10.2 Schema Discovery

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

Every row in a stream or window must have the same structure, or schema, which includes the column names, the column 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 ESP 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 datasource to which your adapter connects. For example, you create a table in your SAP HANA database and 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 ESP 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 particular attention to the data types that the CCL columns inherit from the external datasource. For example, whenever possible, discovery maintains the same level of precision or greater when mapping source data types to ESP data types. Some databases, such as SAP IQ, support microsecond precision for the `SQL_TIMESTAMP` and `SQL_TYPE_TIMESTAMP` data types. As such, schema discovery maps these types to the ESP data type `bigdatetime`, which also supports microsecond precision. If your ESP project does not require this level of precision, you can, after generating your schema through discovery, modify the schema to use a lower-precision data type, such as `msdate` (millisecond precision).

To discover a schema, first configure the adapter properties. Each adapter that supports schema discovery has unique properties that must be set to enable schema discovery.

Make sure that the Data Service has the ODBC Driver Library set correctly. Always use `streamingdbodbc_lib` for Windows (regardless of version). On other platforms, use `streamingdbodbc64_lib` if you are working with the HANA adapter. If not, the choice of driver library depends on the size of SQLLEN in the driver manager. If SQLLEN is 4 bytes, use `streamingdbodbc_lib`. If SQLLEN is 8 bytes use `streamingdbodbc64_lib`. See the SAP Event Stream Processor: Configuration and Administration Guide for more information.

### Related Information

- **Adapter Support for Schema Discovery** [page 41]
  - Lists all adapters currently available from SAP, whether they support schema discovery, and if so, the properties they use to enable it.

- **Discovering a Schema** [page 60]
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.

### 6.10.3 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 the SAP Event Stream Processor Studio. Also, define a data service using the ESP data services view, and add the adapter to the diagram.

**Context**

In the SAP ESP Authoring perspective:

**Procedure**

1. 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, you need to 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.

<table>
<thead>
<tr>
<th>Adapter State</th>
<th>Available Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>The adapter is not attached to a stream or window.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Create a new named schema. Creates a new named schema and populates it with the schema discovered from the adapter.</td>
</tr>
<tr>
<td>The adapter is already attached to a stream or window.</td>
<td>Apply the schema to the connecting stream or window. Populates the stream or window with the schema discovered from the adapter.</td>
</tr>
<tr>
<td></td>
<td>Create a new named schema. Creates a new named schema and populates it with the schema discovered from the adapter.</td>
</tr>
</tbody>
</table>

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 ESP 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 ESP 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 ESP 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.
Related Information

Schema Discovery [page 59]
Discover external schemas and create CCL schemas, streams, or windows based on the format of the data from the datasource that is connected to an adapter.

Adapter Support for Schema Discovery [page 41]
Lists all adapters currently available from SAP, whether they support schema discovery, and if so, the properties they use to enable it.

6.10.4 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.

Context

In the SAP ESP Authoring perspective:

Procedure

1. 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 - 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 ESP which is made up of the RFC schema elements.

7. Click Next.

8. From the left hand side 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 as this is not included.
- For Generic RFC mode, edit the generated mapping file by adding the <variables> and <input> elements as they are not included.
6.10.5 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.

**Context**

In the SAP ESP Authoring perspective:

**Procedure**

1. 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 ESP 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.

6.10.6 Adding an Input Stream or Window to a Project

Input streams and windows accept data from a source external to the 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 as needed, and specify if they need an autogenerate clause. If an autogenerate clause is added, it can be used to automatically generate data for specified columns.

Procedure

1. In the Visual editor workspace, in the Palette menu under the Streams and Windows category, 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 Text Editor, a third duplicate of the original stream or window is created. You can see this third duplicate only when you switch back to the Visual Editor. To remove this third duplicate in the Visual Editor, click Remove all shapes from diagram to clear out all the shapes, then click Add all shapes to get the original stream or window and the second duplicate stream or window back.
4. Click Add Column to add each new column to the schema, then set key columns and edit column expressions.
5. 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 datatype of Long will populate the Candidate list) and click Add.
You can also manually specify a column you want to add to the autogenerate list by clicking *Add Column* and entering in a column name. Only columns with a datatype of Long can be used.

**9.** 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 datatype of Long.

**11.** Click *OK* when finished.

**Related Information**

**Specifying a Retention Policy** [page 66]

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

**Editing Column Expressions** [page 84]

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

**Setting an Aging Policy** [page 115]

An aging policy can be set to flag records that have not been updated within a defined interval. This is useful for detecting records that may be "stale".

### 6.10.7 Specifying a Retention Policy

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

You can set a keep policy, also called a 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.

**Table 4: Keep Policy Options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All rows</td>
<td>Retain all rows in the window (default).</td>
</tr>
<tr>
<td>Last row</td>
<td>Retain only the last row in the window.</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 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:  

```ccl  
DECLARE  
  integer test := 50;  
end;  
```

Then go back and select it. |
| 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 will only occur once 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. |
### Options

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.

**i Note**

The SLACK value cannot be used with the 'Every' option, and thus cannot be used in a Jumping Window count-based retention policy.
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

Creating and Modifying Aggregate Queries [page 73]
   Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

Creating and Modifying Join Queries [page 75]
   Produce a simple query that combines fields from multiple input events into a single output event.

6.10.8 Simple Queries

Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

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 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 SQL UNION operator.

Pattern lets you watch for patterns of events within a single stream or window or across multiple streams and windows. When ESP detects an event pattern in a running project, it produces an output event. This uses the CCL MATCHING clause.

Table 5: CCL Equivalents for Simple Queries (Summary)

<table>
<thead>
<tr>
<th>Simple Query</th>
<th>CCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>WHERE clause</td>
</tr>
<tr>
<td>Aggregate</td>
<td>GROUP BY clause</td>
</tr>
<tr>
<td>Join</td>
<td>FROM clause, WHERE clause, ON clause</td>
</tr>
<tr>
<td>Compute</td>
<td>Simple SELECT statement, with column expressions</td>
</tr>
<tr>
<td>Union</td>
<td>UNION clause</td>
</tr>
<tr>
<td>Pattern</td>
<td>MATCHING clause</td>
</tr>
</tbody>
</table>

Simple Queries from CCL Statements

If you create queries in CCL and want them to appear as simple query shapes in the Visual editor, you must insert a comment immediately preceding the CREATE STREAM, CREATE WINDOW, or CREATE DELTA STREAM statement, in the form:

```ccl
/**@SIMPLEQUERY=<QUERY_TYPE>*/
```

where `<QUERY_TYPE>` is the shape name in the Visual editor.

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 object is a stream, the filter object creates a stream. If the source is a window, delta stream, or flex stream, the filter object creates a window.
- All aggregate objects create a window.
- If the input for a compute object is a stream, the compute object creates a stream. If the source is a window, delta stream, or flex stream, the compute object creates a window.
- If a join object takes input only from streams, then the join object creates a stream. If the source is from one or more windows, delta streams, or flex streams, then the join object creates a window. In a stream-window join, the join object creates a stream.
- If the input of a union object is a stream, the union object creates a stream. If the source is a window, delta stream, or flex stream, the union object creates a window.
- All pattern objects create a stream.

**Note**

Delta streams are supported only in CCL: you cannot create them in Studio. SAP recommends delta streams for advanced users. Keyed streams support some of the same use cases as delta streams and are easier to work with.

Related Information

**Shape Reference** [page 49]

*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.*

**Adding Shapes to a Diagram** [page 54]

*Create streams, windows, and shared components, relate them using continuous queries, and attach them to adapters.*

**Connecting Elements** [page 82]

*Connect two shapes in a diagram to create a data flow or link between them.*

**Queries in CCL** [page 137]

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

**Creating and Modifying Filter Queries** [page 72]

*Produce a simple query that only passes on events with specific characteristics. Filter uses a CCL WHERE clause.*

**Creating and Modifying Aggregate Queries** [page 73]

*Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.*

**Creating and Modifying Compute Queries** [page 74]

*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.*

**Creating and Modifying Join Queries** [page 75]
Produce a simple query that combines fields from multiple input events into a single output event.

Creating and Modifying Union Queries [page 80]
Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Creating and Modifying Pattern Queries [page 81]
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.

6.10.8.1 Creating and Modifying Filter Queries

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, 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 the appropriate stream or window.
   Attach filter objects to any stream, window, or Flex operator. Filter objects can have only one input.
4. To edit the value of the filter expression, select the value and change 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.

Related Information

Creating and Modifying Aggregate Queries [page 73]
Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

Creating and Modifying Compute Queries [page 74]
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.

Creating and Modifying Join Queries [page 75]
Produce a simple query that combines fields from multiple input events into a single output event.

Creating and Modifying Union Queries [page 80]
Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Creating and Modifying Pattern Queries [page 81]
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.
Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

### 6.10.8.2 Creating and Modifying Aggregate Queries

Produce a simple query that combines data, similar to the CCL `GROUP BY`, `GROUP FILTER`, and `GROUP ORDER` clauses.

**Procedure**

1. In the Visual editor Palette, in `Streams and Windows`, select `Aggregate`.
2. Select a location in the diagram and click to add the shape.
3. 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.
4. Add columns:
   - Click `Copy Columns from Input` in the shape toolbar to select the columns to copy into the schema for the Aggregate window.
   - Add additional columns by clicking `Add Column Expression` in the shape toolbar.
   - 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 $\text{Ctrl}+\text{F2}$ to edit it using the pop-up expression editor.

   **Tip**
   
   When entering column names and their datatypes, use $\text{Tab}$ to easily move between cells in the table.

5. 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.

6. (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.

7. (Optional) Use the Toggle option to designate the aggregate object as LOCAL or OUTPUT. By default, aggregates are Output.
Related Information

Creating and Modifying Filter Queries [page 72]
   Produce a simple query that only passes on events with specific characteristics. Filter uses a CCL \texttt{WHERE} clause.

Creating and Modifying Compute Queries [page 74]
   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.

Creating and Modifying Join Queries [page 75]
   Produce a simple query that combines fields from multiple input events into a single output event.

Creating and Modifying Union Queries [page 80]
   Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Creating and Modifying Pattern Queries [page 81]
   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 \texttt{MATCHING} clause.

Simple Queries [page 69]
   Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

Specifying a Retention Policy [page 66]
   The keep policy determines the basis for retaining rows in a window.

6.10.8.3 Creating and Modifying Compute Queries

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.

Procedure

1. In the Visual editor Palette, in \texttt{Streams and Windows}, select \texttt{Compute} \(\text{Compute}\).  
2. Select a location in the diagram and click to add the shape.
3. 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.
4. Add columns:
   a. Click \texttt{Copy Columns from Input} \(\text{Copy Columns from Input}\) in the shape toolbar to copy input fields into the schema for this query.
   b. Add additional columns by clicking \texttt{Add Column Expression} \(\text{Add Column Expression}\) 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 \texttt{Ctrl}+\texttt{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.

5. Add column expressions as necessary.
6. Modify column expressions by selecting and modifying them directly, or by editing the corresponding fields in the Properties view.
7. Use the toggle option to designate the compute object as INPUT or OUTPUT. By default, computes are Output.

Related Information

Creating and Modifying Filter Queries [page 72]
Produce a simple query that only passes on events with specific characteristics. Filter uses a CCL WHERE clause.

Creating and Modifying Aggregate Queries [page 73]
Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

Creating and Modifying Join Queries [page 75]
Produce a simple query that combines fields from multiple input events into a single output event.

Creating and Modifying Union Queries [page 80]
Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Creating and Modifying Pattern Queries [page 81]
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.

Simple Queries [page 69]
Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

6.10.8.4 Creating and Modifying Join Queries

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.
Streams, windows and delta streams can participate in a join. However, a delta stream may participate in a join only if it has a **KEEP** clause specified. Only one stream can participate in a join. For details of supported joins, see the *SAP Event Stream Processor: Developer Guide*.

**Tip**

To add multiple connections, *Shift+click* and hold the **Connector** tool and add connections. 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 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.

   Or, edit the corresponding fields in the Properties view.

   **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.

   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

*Creating and Modifying Filter Queries* [page 72]

> Produce a simple query that only passes on events with specific characteristics. *Filter uses a CCL WHERE clause.*

*Creating and Modifying Aggregate Queries* [page 73]

> Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

*Creating and Modifying Compute Queries* [page 74]
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.

Creating and Modifying Union Queries [page 80]

Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Creating and Modifying Pattern Queries [page 81]

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.

Simple Queries [page 69]

Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

Specifying a Retention Policy [page 66]

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

Join Types and Restrictions [page 77]

Determine what combination of attributes your join simple query must contain.

6.10.8.4.1 Join Types and Restrictions

Determine what combination of attributes your join simple query must contain.

In order to determine what type of join simple query you want to create in the ESP Studio, use this reference to determine how components of your join can be attached, and what settings to modify in the Edit Join Expression dialog box.

**Note**

If you have created a join using comma-separated syntax in the CCL editor, and subsequently added 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.

Streams, windows, or delta streams can participate in a join. However, a delta stream can participate in a join only if it has a keep policy defined. A join can contain any number of windows and delta streams (with their respective keep policies), but only one stream. Self joins are also supported. For example, you can include the same window or delta stream 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 columns in aggregation automatically forms the key for the target window. This restriction does not apply to delta stream-window joins because use of the keep policy converts a delta stream into an unnamed window.

SAP Event Stream Processor supports all join types:

<table>
<thead>
<tr>
<th>Join Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner Join</td>
<td>INNER JOIN</td>
<td>One record from each side of the join is required for the join to produce a record.</td>
</tr>
</tbody>
</table>
### Join Type

<table>
<thead>
<tr>
<th>Join Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Outer Join</td>
<td>LEFT JOIN</td>
<td>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.</td>
</tr>
<tr>
<td>Right Outer Join</td>
<td>RIGHT JOIN</td>
<td>Reverse of left outer join, where the right side is the outer side and the left side is the inner side of the join.</td>
</tr>
<tr>
<td>Full Outer Join</td>
<td>FULL JOIN</td>
<td>A record is produced whether there is a match on the right side or the left side of the join.</td>
</tr>
</tbody>
</table>

SAP Event Stream Processor also supports these cardinalities:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-One</td>
<td>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.</td>
</tr>
<tr>
<td>One-Many</td>
<td>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.</td>
</tr>
<tr>
<td>Many-Many</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### 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.

### Key Field Rules

Key field rules are necessary to ensure that rows are not rejected due to duplicate inserts or due to the key fields being NULL. Because regular streams do not use primary keys, these rules apply only to windows and delta streams.

- 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.

This table illustrates this information in the context of join types:

<table>
<thead>
<tr>
<th></th>
<th>One-One</th>
<th>One-Many</th>
<th>Many-One</th>
<th>Many-Many</th>
</tr>
</thead>
<tbody>
<tr>
<td>INNER</td>
<td>Keys from at least one side should be included in the projection list (or a combination of them if keys are composite).</td>
<td>Keys from the right side should be included in the projection list.</td>
<td>Keys from the left side should be included in the projection list.</td>
<td>Keys from both sides should be included in the projection list.</td>
</tr>
<tr>
<td>LEFT</td>
<td>Keys from the left side alone should be included.</td>
<td>Not allowed.</td>
<td>Keys from the left side should be included in the projection list.</td>
<td>Not allowed.</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Keys from the right side alone should be included.</td>
<td>Keys from the right side should be included in the projection list.</td>
<td>Not allowed.</td>
<td>Not allowed.</td>
</tr>
<tr>
<td>OUTER</td>
<td>Keys should be formed using firstnonnull () on each pair of keys from both sides.</td>
<td>Not allowed.</td>
<td>Not allowed.</td>
<td>Not allowed.</td>
</tr>
</tbody>
</table>

These options can be defined in the Options pane of the Edit Join Expression dialog box.

**Nested Joins**

Several important functions are necessary to note in SAP Event Stream Processor when implementing a nested join. 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.

**Related Information**

Creating and Modifying Join Queries [page 75]

*Produce a simple query that combines fields from multiple input events into a single output event.*
6.10.8.5 Creating and Modifying Union Queries

Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Procedure

1. In the Visual editor Palette, 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.

Related Information

Creating and Modifying Filter Queries [page 72]
Produce a simple query that only passes on events with specific characteristics. Filter uses a CCL WHERE clause.

Creating and Modifying Aggregate Queries [page 73]
Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

Creating and Modifying Compute Queries [page 74]
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.

Creating and Modifying Join Queries [page 75]
Produce a simple query that combines fields from multiple input events into a single output event.

Creating and Modifying Pattern Queries [page 81]
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.

Simple Queries [page 69]
Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.
6.10.8.6 Creating and Modifying Pattern Queries

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. In the Visual editor Palette, in Streams and Windows, click Pattern.
2. Select a location in the diagram and click to add the shape.
3. Connect the Pattern shape to one or more streams or windows that are the inputs to query.
4. 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.
   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.

5. 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.

   For details of valid pattern expressions, see **ON Clause: Pattern Matching Syntax** in the SAP Event Stream Processor: Developer Guide.

6. (Optional) Use the toggle option to designate the pattern object as LOCAL or OUTPUT. By default, patterns are Output.

**Related Information**

- Creating and Modifying Filter Queries [page 72]
  Produce a simple query that only passes on events with specific characteristics. Filter uses a CCL `WHERE` clause.
- Creating and Modifying Aggregate Queries [page 73]
Produce a simple query that combines data, similar to the CCL GROUP BY, GROUP FILTER, and GROUP ORDER clauses.

Creating and Modifying Compute Queries [page 74]

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.

Creating and Modifying Join Queries [page 75]

Produce a simple query that combines fields from multiple input events into a single output event.

Creating and Modifying Union Queries [page 80]

Use a union object to combine two or more input streams or windows into a single output. All inputs must have matching schema.

Simple Queries [page 69]

Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

6.10.9 Connecting Elements

Connect two shapes in a diagram to create a data flow or link between them.

Context

The Connector tool creates flows between streams and windows, establishes references between streams and shared components, or attaches notes between shapes.

Procedure

1. In the Palette view, select the Connector tool.
2. Click the shape that will produce the output.
   This attaches the connector line to the first shape.
3. Click the shape that will receive the data to indicate the direction of data flow.

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Arrow]</td>
<td>Connection is allowed</td>
</tr>
<tr>
<td>![Crossed Arrow]</td>
<td>Connection is not allowed</td>
</tr>
</tbody>
</table>
Tip

To add multiple connections, **Shift+click** and hold the **Connector** tool and add connections. To return to normal selection, press **Esc** or click the **Select** tool in the Palette to release it.

---

Related Information

**Simple Queries** [page 69]

Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

**Shape Reference** [page 49]

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.

6.10.10 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 are displayed as 🔒 icons. Deduced keys 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. 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. 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. 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.

6.10.11 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, either:
   ○ 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

Column Expressions [page 85]

A column expression produces a result based on the value of input columns, the relationship of column values to each other, or the computed formulas. It may include built-in or user-defined functions, constants, parameters, or variables.
6.10.11.1 Column Expressions

A column expression produces a result based on the value of input columns, the relationship of column values to each other, or the computed formulas. It may include built-in or user-defined functions, constants, parameters, or variables.

Simple Expressions

A simple CCL expression specifies a constant, NULL, or a column. A constant can be a number or a text string. The literal NULL denotes a null value. NULL is never part of another expression, but NULL by itself is an expression.

To specify a column, include both the column name and the stream or window name, using the format `source.column-name`.

Some valid simple expressions include:
- `stocks.volume`
- `'this is a string'`
- `26`

Compound Expressions

A compound CCL expression is a combination of simple or compound expressions. Compound expressions can include operators and functions, as well as the simple CCL expressions (constants, columns, or NULL).

You can use parentheses to change the order of precedence of the expression’s components.

Some valid compound expressions include:
- `sqrt (9) + 1`
- `('example' + 'test' + 'string')`
- `( length ('example') *10 ) + pi()`

Column Alias in Expressions

Each expression defines a unique name or alias for the column.

In the PortfolioValuation example, 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:

<table>
<thead>
<tr>
<th>Alias</th>
<th>Column Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>PriceFeed.Symbol</td>
</tr>
<tr>
<td>Alias</td>
<td>Column Expression</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
</tr>
<tr>
<td>LastPrice</td>
<td>PriceFeed.Price</td>
</tr>
<tr>
<td>VWAP</td>
<td>( ( \text{sum} \left( \left( \text{PriceFeed_Price} \times \text{CAST} \left( \text{FLOAT}, \text{PriceFeed_Shares} \right) \right) \right) / \text{CAST} \left( \text{float}, \text{sum} \left( \text{PriceFeed_Shares} \right) \right) ) )</td>
</tr>
<tr>
<td>LastTime</td>
<td>PriceFeed.TradeTime</td>
</tr>
</tbody>
</table>

**Datatypes in Expressions**

Datatypes for column expressions are inherited from the schema, either an explicitly created inline schema, or one discovered from the input adapter. You choose from supported datatypes in the schema editor, not in the column expression editor.

Enclose string data in expressions in single quotes, for example, 'my_string_data'.

**Case Sensitivity**

- All identifiers are case sensitive. This includes names of streams, windows, parameters, variables, schemas, and columns.
- Keywords are case insensitive, and cannot be used as identifier names.
- Built-in function names (except keywords) and user-defined functions are case sensitive, however, some built-in function names have both lowercase and mixed case forms, for example, `setOpcode` and `setopcode`.

**Related Information**

- **CCL Functions** [page 138]
  
  A function is a self-contained, reusable block of code that performs a specific task.

- **Operators** [page 139]
  
  CCL supports a variety of numeric, nonnumeric, and logical operator types.

- **Editing Column Expressions** [page 84]
  
  Modify column expressions for windows and streams using an inline editor or dialog-based expression editor.
6.10.12 Deleting an Element

Delete an element from the project to remove it completely, or delete it from the diagram only.

Procedure

1. Select one or more elements in the diagram.
2. Right-click and choose either:
   ○ *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.
3. When you choose *Delete Element*, confirm the deletion.

Related Information

Adding Shapes to a Diagram [page 54]
*Create streams, windows, and shared components, relate them using continuous queries, and attach them to adapters.*

Keyboard Shortcuts in the Visual Editor [page 55]
*Use keyboard shortcuts to access various functions quickly within the Visual editor.*

6.11 Adding Advanced Features to a Project

Complete your project by adding more complex operations and expressions, reusable modules and named schemas, and custom adapters.

Context

All of these advanced features are optional.
6.11.1 Complex Queries

Use the generic derived stream, and derived window shapes to create more complex continuous queries in the Visual editor than the ones you can create with the simple query shapes.

A derived stream, derived window, or derived delta stream takes input from another stream or window, rather than directly from an adapter, and applies a continuous query to it. All of the simple queries in the Visual editor are a type of derived stream or derived window.

For example, to create a continuous query that applies both a set of join conditions and a pattern matching expression, use a generic derived window.

Choose the shape type according to your input, output, and retention requirements for data, and for preserving insert, update, and delete operations.

Table 6: Derived Stream, Derived Window, and Derived Delta Stream Rules

<table>
<thead>
<tr>
<th>Shape</th>
<th>Input</th>
<th>Output</th>
<th>Retains state</th>
<th>Preserves inserts, updates, and deletes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived Stream</td>
<td>Another stream</td>
<td>Stream</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Derived Window</td>
<td>Another stream or window</td>
<td>Window</td>
<td>As defined in Keep policy (default is keep all rows)</td>
<td>yes</td>
</tr>
<tr>
<td>Derived Delta Stream</td>
<td>Another window or delta stream</td>
<td>Stream</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Note
- In order to derive a window from a stream, a GROUP BY clause must be included in the query.
- A derived delta stream can only be created in CCL using the CREATE DELTA STREAM statement.

Related Information

Join Types and Restrictions [page 77]
- Determine what combination of attributes your join simple query must contain.

Operation Codes [page 12]
- The operation code (opcode) of an event record specifies the action to perform on the underlying store of a window for that event.

Editing Column Expressions [page 84]
Modify column expressions for windows and streams using an inline editor or dialog-based expression editor.

6.11.2 Reference Table Queries

Reference table queries enable you to look up information in a 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 a 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, you must first decide 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, you need to decide what you want as the output of the join. There are numerous restrictions on how you can 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 must 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 utilize a reference in the same ways you can utilize 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

You must have the SAP HANA ODBC client installed on the system where you want to run projects that include reference table queries. SAP recommends that you use the latest version of the SAP HANA ODBC client available, but it must be at least version 1.0.73.

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 ESP Authoring 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 a SAP HANA cluster, refer to Configuring to Support SAP HANA Failover in the SAP Event Stream Processor: Configuration and
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. Turning on caching for a reference table query enables local storage of previous query results. Thus, 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). Caching can be turned 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. This behavior may not always be desired.

In addition to turning caching on and off, 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, it 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.)

In order to check the type for each mapped column, the reference attempts to pull a sample row from the database. It’s done this way to be as database-agnostic as possible. If it can pull that column into ESP, the check succeeds. Otherwise it fails, except in the following two 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 will not complete type checking, but will allow 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, then the type checking will stop, and a warning will be 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 will attempt to reconnect based on the reconnect parameters specified in the reference’s definition.
6.11.2.1 Creating a Reference Table Query

Create a reference in a Studio project that queries a table in an SAP HANA database.

Procedure

1. In the SAP ESP Authoring perspective - in the Palette under Streams and Windows - select Reference.
2. Select a location in the diagram and click to add the shape.
3. Provide a name for the Reference.
4. 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 will be prompted to do so.
5. 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.
6. 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.
7. 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.
8. In the Cache Policy field, select <ONACCESS> to turn on caching, or leave the default value, <NONE>, to leave caching turned off.
9. If you selected <ONACCESS> for the Cache Policy,
   ○ in the Max Cache Size(MB) field, you may enter the maximum amount, in megabytes, of memory that you want to allow the cache to use, or leave it blank to allow an unlimited cache size.
   ○ in the Max Cache Age field, you may enter the interval (for example, 5 minutes) for which you want a query response to be retrievable from the cache, or leave it blank to allow using cached results no matter how old they are.

   **Note**

   To decide whether or not 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. When deciding on the amount of time to keep query results available in the cache, you should also consider how often the information changes and how important it is to get current information.

10. 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.
11. Click OK.

By default, ESP 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 found in the SAP HANA database table the reference queries.

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 and Editing a Module.

6.11.2.2 Using a Reference Table Query in a Join

Use a join to combine incoming event data and source data from a reference table query.

Procedure

1. Obtain the necessary information about the SAP HANA 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. A stream and a reference in a join only produces a stream; a window and a reference in a join produces a window, provided that the primary keys are set in the reference and input window.
4. Create a join to combine the incoming event data from the input stream or window with the data obtained from the reference table query:
   a. From the Palette, under Streams and Windows, select Join, then click an empty area in the diagram.
   b. From the Palette, select Connector and connect the input stream or window to the join.
   c. From the Palette, select Connector and connect the reference query table to the join.
6.11.2.3 Using a Reference Table Query in a Flex

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 the SAP Event Stream Processor: Developer Guide for examples of CCLScript routines in a Flex operator.

6.11.3 Modularity

A module in SAP Event Stream Processor offers reusability; it can be loaded and used multiple times in a single project or in many projects.

Modularity means organizing project elements into self-contained, reusable components called 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 accessible within that scope only; they are inaccessible 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, however, are invalid. 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.

The CREATE MODULE statement creates a module that can be loaded multiple times in a project, where its inputs and outputs can be bound to different parts of the larger project. The LOAD MODULE statement allows reuse of a defined module one or more times throughout a project. Modularity is particularly useful when used with the IMPORT statement, which allows you to use (LOAD) modules created in a separate CCL file.

**Note**

All module-related compilation errors are fatal.

### 6.11.3.1 Creating a Module

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. In the Visual editor Palette, in Shared Components, select Module ( ).
2. 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 will open a new diagram where you can add input streams/windows, simple queries, and derived streams/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.
6.11.3.2 Editing a Module

Edit basic module properties and module input, output, and import functions.

Prerequisites

Create the 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. In the Visual editor, select the module to edit.
2. To edit the module name to be unique across all object names in the scope for this module, either:
   ○ 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 ESP Authoring perspective.

3. Click **Add Module Exported Reference(s)**.

4. In the Module Exported Reference(s) dialog, select the reference(s) to add or remove, then click **OK**.

5. Click **Add Module Inputs**.

6. In the Module Inputs dialog, select the inputs to add or remove, then click **OK**.

7. Select **Add Module Outputs**.

8. In the Module Outputs dialog, select the outputs to add or remove, then click **OK**.

9. To access and edit the contents of the **CREATE MODULE** statement, select **Open Module Diagram**.

10. Edit the module in the diagram that opens.

11. Add comments in the Properties view.

---

**Related Information**

- **Creating a Module** [page 94]
  
  Add a new module to an existing project in the Visual editor.

- **Creating a Module File** [page 96]
  
  Create a new, separate module file that can be imported into a project.

- **Importing Definitions from Another CCL File** [page 97]
  
  Import a module file to use the module in your project.

- **Using a Module Within a Project** [page 98]
  
  Create an instance of a defined module within the project, and allow the inputs and outputs of the module to be bound to streams or windows in the project.

- **Configuring a Loaded Module** [page 99]
  
  Add or remove input and output bindings and parameter values (if any) for a specific module instance.

- **Configuring a Module Repository** [page 101]
  
  Create a folder in which to store modules and configure the Studio to use it.

---

**6.11.3.3 Creating a 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 that separately hold a `CREATE MODULE` statement.
Procedure

1. Choose **File ➤ New ➤ CCL Module File**.
2. Enter a file name. This becomes the module name, and must be 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 Event Stream Processor: Developer Guide*.

Related Information

- **Creating a Module** [page 94]
  Add a new module to an existing project in the Visual editor.
- **Editing a Module** [page 95]
  Edit basic module properties and module input, output, and import functions.
- **Importing Definitions from Another CCL File** [page 97]
  Import a module file to use the module in your project.
- **Using a Module Within a Project** [page 98]
  Create an instance of a defined module within the project, and allow the inputs and outputs of the module to be bound to streams or windows in the project.
- **Configuring a Loaded Module** [page 99]
  Add or remove input and output bindings and parameter values (if any) for a specific module instance.
- **Configuring a Module Repository** [page 101]
  Create a folder in which to store modules and configure the Studio to use it.

6.11.3.4 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 Outline view in the Visual editor, as described here.
**Procedure**

1. Select the **SAP ESP Authoring** tab.
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.

**Related Information**

- **Creating a Module** [page 94]
  Add a new module to an existing project in the Visual editor.

- **Editing a Module** [page 95]
  Edit basic module properties and module input, output, and import functions.

- **Creating a Module File** [page 96]
  Create a new, separate module file that can be imported into a project.

- **Using a Module Within a Project** [page 98]
  Create an instance of a defined module within the project, and allow the inputs and outputs of the module to be bound to streams or windows in the project.

- **Configuring a Loaded Module** [page 99]
  Add or remove input and output bindings and parameter values (if any) for a specific module instance.

- **Configuring a Module Repository** [page 101]
  Create a folder in which to store modules and configure the Studio to use it.

**6.11.3.5 Using a Module Within a Project**

Create an instance of a defined module within the 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. 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 no CREATE_MODULE statements are found, the Palette drawer is empty.
2. Click anywhere in the diagram to place the load module.

Related Information

Creating a Module [page 94]
Add a new module to an existing project in the Visual editor.

Editing a Module [page 95]
Edit basic module properties and module input, output, and import functions.

Creating a Module File [page 96]
Create a new, separate module file that can be imported into a project.

Importing Definitions from Another CCL File [page 97]
Import a module file to use the module in your project.

Configuring a Loaded Module [page 99]
Add or remove input and output bindings and parameter values (if any) for a specific module instance.

Configuring a Module Repository [page 101]
Create a folder in which to store modules and configure the Studio to use it.

6.11.3.6 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. In the diagram, select the module instance to edit.
2. To edit the name of the module instance, either:
   ○ Click the load module instance name.
   ○ In verbose mode, click Edit.
3. Set the input bindings by adding connectors: first expand the Input Bindings compartment to that you can see the list of inputs. Then 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 will open the model in a separate editor so that you can see the structure of the module.

4. Output bindings will have been set automatically, and the outputs will appear on the diagram attached to the module instance. You can rename the outputs as desired. Note: for input bindings the schema on both sides of the binding needs to be compatible.

5. Further modify input or output bindings by selecting an individual binding in the load module, and changing any of these options in the Properties window:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputStreamOrWindow</td>
<td>Select the available input stream or window components from the list.</td>
</tr>
<tr>
<td>streamOrWindowInModule</td>
<td>Select the available stream or window to bind with existing stream or window inputs.</td>
</tr>
<tr>
<td>comment (Output only)</td>
<td>Add a comment or description of the output stream.</td>
</tr>
<tr>
<td>name (Output only)</td>
<td>Add a name to the output stream.</td>
</tr>
</tbody>
</table>

6. If the module uses any parameters, Parameter bindings will be 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.

7. (Optional) Click Add Store Binding ( ). If you omit a store binding, the default memory store will be used. You can optionally specify a store for windows in the module.

8. 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

9. To access input or output windows used inside a load module, select Open Module Diagram ( ).

**Related Information**

Creating a Module [page 94]
Add a new module to an existing project in the Visual editor.

Editing a Module [page 95]
Edit basic module properties and module input, output, and import functions.

Creating a Module File [page 96]
Create a new, separate module file that can be imported into a project.

Importing Definitions from Another CCL File [page 97]
Import a module file to use the module in your project.

Using a Module Within a Project [page 98]
Create an instance of a defined module within the project, and allow the inputs and outputs of the module to be bound to streams or windows in the project.

Configuring a Module Repository [page 101]
Create a folder in which to store modules and configure the Studio to use it.

6.11.3.7 Configuring a Module Repository

Create a folder in which to store modules and configure the 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 Studio Palette.

Procedure

1. Create a new folder, or select an existing folder, to serve as the module repository.
2. In Studio, click Windows Preferences SAP Event Stream Processor.
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.

Related Information

Creating a Module [page 94]  
Add a new module to an existing project in the Visual editor.

Editing a Module [page 95]  
Edit basic module properties and module input, output, and import functions.

Creating a Module File [page 96]  
Create a new, separate module file that can be imported into a project.

Importing Definitions from Another CCL File [page 97]  
Import a module file to use the module in your project.

Using a Module Within a Project [page 98]  
Create an instance of a defined module within the project, and allow the inputs and outputs of the module to be bound to streams or windows in the project.

Configuring a Loaded Module [page 99]  
Add or remove input and output bindings and parameter values (if any) for a specific module instance.
6.11.4 Stores

Set store defaults, or choose a log store or memory store to determine how data from a window is saved.

Every window is assigned to a store, which holds the retained records. By default, all windows are assigned to a memory store. Additional stores can be created to add data recoverability and to optimize performance. Windows can then be assigned to specific stores.

You can also create a default store explicitly with the `CREATE DEFAULT MEMORY STORE` and `CREATE DEFAULT LOG STORE` statements. By stipulating default store settings you can determine store types and locations in the event that you do not assign new windows to specific store types.

Log Stores

The log store holds all data in memory, but also logs all data to the disk, meaning it guarantees data state recovery in the event of a failure. Use a log store to be able to recover the state of a window after a restart.

Log stores are created using the `CREATE LOG STORE` statement. You can set a log store as the default store using the `CREATE DEFAULT LOG STORE` statement, which overrides the default memory store.

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.

Memory stores are created using the `CREATE MEMORY STORE` statement. If no default store is defined, new windows are assigned to a memory store automatically. You can use either of the relevant statements shown above to determine specific memory store behavior and set default store settings.

6.11.4.1 Creating a Log Store

Create a log store to allow recovery of data in a window in the event of a server shutdown or failure.

Prerequisites

Consult with your system administrator on the size, number, and location of log stores, to ensure optimal performance.
Procedure

1. In the Visual editor Palette, in Shared Components, click Log Store.
2. Select a location in the diagram and click to add the shape.
3. Connect the log store to a window.
4. Click Set Store Properties and modify property values.

Note
The table lists property names first as shown in the Properties dialog, then as shown in the Properties compartment of the store shape.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| File name (FILENAME) | The absolute or relative path to the folder where log store files are written. A relative path is preferred. When sandboxing is disabled, the filename property only creates the directory for the last level specified. For example, if the property is set to C:\dir1\dir2\dir3, the dynamic.log file will be located in C:\dir1\dir2\dir3, provided that C:\dir1\dir2\ already exists and the SAP ESP project has permissions to create dir3. If C:\dir1\dir2\ or permissions do not exist, ESP generates a runtime error and the project stops running. When sandboxing is enabled, absolute paths are changed into relative paths, relative to the sandbox base directory, and interpreted as literally as possible. Both a relative path and absolute path automatically place log stores under: <sandbox-base-directory>/store/<filename-property>/ <workspace-name>.<project-name>.<instance-number>. For Windows, drive letters will be turned into a folder and preserved in the new directory. For example, a sandbox base directory of work1/esp/sandbox with log stores with filename parameters of D:/esp/logstore and data/logstore running in project importantpl in workspace worksp1 instance number 0 will become:  
  - For D:/esp/logstore:  
    - work1/esp/sandbox/store/d/esp/logstore/worksp1.importantpl.0  
  - For data/logstore:  
    - work1/esp/sandbox/store/data/logstore/worksp1.importantpl.0 |
| Max Size (GB) (MAXFILESIZE) | The maximum size of the log store file in MB. Default is 8MB.                                                                                                                                             |
| Sweep Amount (%) (SWEEPAMOUNT) | The amount of data, in megabytes, that can be cleaned in a single pass. Default is 20 percent of maxfilesize.                                                                                           |
| Reserve Percentage (%) (RESERVEPCT) | The percentage of the log to keep as free space. Default is 20 percent.                                                                                                                               |
| Ck Count (CKCOUNT) | The maximum number of records written before writing the intermediate metadata. Default is 10,000.                                                                                                        |
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync (SYNC)</td>
<td>Specifies whether the persisted data is updated synchronously with every stream being updated. A value of true guarantees that every record acknowledged by the system is persisted at the expense of performance. A value of false improves performance, but it may result in a loss of data that is acknowledged, but not yet persisted. Default is false.</td>
</tr>
</tbody>
</table>

5. (Optional) Select **Default** to make this the default store for the project (or module).

#### Related Information

**Creating a Memory Store** [page 112]

Create a memory store to retain the state of continuous queries in memory, from the most recent server startup.

#### 6.11.4.1.1 Log Store Guidelines

Special considerations for using log stores.

**General Guidelines**

- Locate log stores on a shared drive accessible 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 into a log store any window fed by stateless elements (streams and delta streams).
- Put into a log store any window 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.
- Put into a log store any window that 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 it is attached to.
- Variables and CCLScript data structures (dictionaries, vectors, and event caches) do not persist in log stores and thus cannot be recovered after a failure. Use these structures with log stores only when:
  - You can provide logic to reconstruct the structures on restart, or
  - Processing will not be 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:

- Because copies of events are kept in the same log store the stream or window is assigned to, the log store for a guaranteed delivery stream or window must be significantly larger than the log store for a similar stream or window without guaranteed delivery. 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.

- Put into a log store any window on which GD is enabled and all input windows that feed GD windows. 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.

6.11.4.1.2 Sizing a Log Store

Calculate the size of the log store your project requires. Correctly sizing your log store is important, as stores that are too small or large can lead to performance issues.

Prerequisites

Review the log store guidelines.

Context

You will 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 step 1 [page 107]:

\[\text{HeaderSize}(56) + \text{Offsets}(4 \times M) + \sum_1^M \text{PS}\]

In the formula,

- \(M\) represents the number of columns
- \(\text{PS}\) 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. This table gives the size for datatype.

Table 8: Primitive Datatype Sizes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Size in Bytes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>1 + number of characters in the string</td>
<td>Estimate an average length</td>
</tr>
<tr>
<td>Float</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Money(n)</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>SecondDate</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>MsDate</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>BigDateTime</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td>4 + number of bytes in the binary value</td>
<td>Estimate an average length</td>
</tr>
</tbody>
</table>

**Note**

Guaranteed delivery (GD) logs hold events stored for delivery. If no GD logs are stored in the log store, you have the option of skipping step 1 [page 107], step 2 [page 108], and step 3 [page 108]. 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 the SAP Event Stream Processor: Studio Users Guide for details on Playback or the SAP Event Stream Processor: Utilities Guide for details on streamingplayback.) The log store reports "liveSize" in the project log file (esp_server.log) 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 [page 108] and beyond.

**Procedure**

1. 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:

```sql
CREATE SCHEMA TradesSchema AS {
  TradeId   LONG,
  Symbol    STRING,
  Price     MONEY(4),
  Volume    INTEGER,
  TradeDate BIGDATETIME
};
```
a. Using the primitive sizes from the Table 8: Primitive Datatype Sizes [page 107] table, 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:

\[ 8 + (4 + 1) + 8 + 4 + 8 = 33 \text{ bytes} \]

b. Add the size of the offsets to the size of the column values. The offsets are calculated as \((4 \times M)\) where \(M\) is the number of columns. Plugging in the sample schema’s five columns, we get:

\[ (4 \times 5) + 33 = 53 \text{ bytes} \]

c. Add the size of the row header, which is always 56 bytes:

\[ 56 + 53 = 113 \text{ bytes} \]

d. Round up to the nearest number divisible by:
\[ 8 \] if ESP is running on a 64-bit architecture
\[ 4 \] if ESP is running on a 32-bit architecture

For a 64-bit installation, use this formula:

\[ \text{URS} + (8 - (\text{URS modulo 8})) \]

where \(\text{URS}\) is the unrounded record size value you computed in step 1.c [page 108]. (For a 32-bit installation, substitute a 4 for each 8 in the formula.) Continuing with our example, where we assume ESP is running on a 64-bit machine,

\[ 113 + (8 - (1)) = 120 \text{ bytes} \]

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 you must 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, and 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 because GD logs are purged only when there are at least 1000 fully committed events.

Next, for each window, determine the data size by multiplying the expected record count by the recordSize you computed in step 1.e [page 108]. 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.

Also sum the record counts for each window and GD log assigned to this log store and label that value recordCount.

3. To calculate the basic indexing overhead, multiply the recordCount from step 2 [page 108] by 96 bytes. Add the result to the dataSize value.

4. Choose the value of the reservePct parameter. The required store size, in bytes, including the reserve, is calculated as:

\[ \text{storeBytes} = \text{dataSize} \times 100 / (100 - \text{reservePct}) \]

where \(\text{dataSize}\) is the value you computed in step 3 [page 108]. 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. The records in the queues that are located early in the sequence
must be counted together with any records they produce as they are processed through the project. Include the number of output records that are 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

- Determine the number of records that are 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 from derived1). Assuming that derived2 is a join and generates on average 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.

When the project’s topology is not linear, you must take all branches into account. The pass-through ratio may be different for data coming from the different parent streams. You must 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.

- 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.

- 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:
  \[\text{uncheckpointedBytes} < \text{storeBytes} \times (\text{reservePct} / 4) / 100\]
  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.

When you create the log store, place storeBytes, the log store size value you arrive at here, in the `CREATE LOG STORE` statement’s maxfilesize parameter.
6.11.4.1.3 Log Store Sizing Reference

Set sizing parameters for a log store in a `CREATE LOG STORE` statement in the project’s CCL file. 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. See Sizing a Log Store for instructions on calculating this value.

Unlike memory stores, log stores do not extend automatically. Sizing 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 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. It 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 30MB, 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 Event Stream Processor 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 must be 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 this point, the source streams are temporarily stopped, the streams quiesced, and the checkpoint and cleaning cycle are performed. Streams do not quiesce immediately: they 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, the store becomes wedged, because it cannot perform the cleaning cycle. The automatic reserve calculation does not account for uncheckpointed data.
Log Store Size Warnings

As the amount of data in the store grows, if the free space falls below 10 percent (excluding the reserve), Event Stream Processor starts reporting "log store is nearing capacity" in the project log file (esp_server.log). 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 Event Stream Processor when convenient, and increase the store size. Otherwise, Event Stream Processor 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 Event Stream Processor 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:

$$96 + 32 \times \text{ceiling}\left(\log_2(\text{number\_of\_records\_in\_the\_stream})\right)$$

If a stream is small (for example, fewer than 1000 records), the overhead for each record is:

$$96 + 32 \times \text{ceiling}\left(\log_2(1000)\right) = 96 + 32 \times 10 = 416$$

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 file is “swept through” during each cleaning pass. It must be 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, SAP recommends that you 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 \texttt{vmstat} to display high disk usage due to system paging.

For storing data locally using log stores, SAP recommends that you 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.

6.11.4.2 Creating a Memory Store

Create a memory store to retain the state of continuous queries in memory, from the most recent server startup.

Prerequisites

Consult with your system administrator on the type, number, and index values for memory stores, to ensure optimal performance.
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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Size Hint (KB) (INDEXSIZE-HINT)</td>
<td>(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.</td>
</tr>
<tr>
<td>Index Kind (INDEXTYPE)</td>
<td>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.</td>
</tr>
</tbody>
</table>

6. (Optional) Select Default to make this the default store for the project (or module).

Related Information

Creating a Log Store [page 102]
Create a log store to allow recovery of data in a window in the event of a server shutdown or failure.

6.11.5 Flex Operators

Flex operators are custom operators that let you write CCLScript scripts to operate on incoming events.

Flex operators extend the type of business logic that can be applied to incoming events, beyond what you can do with standard CCL or SQL queries. They extend CCL by allowing you to write individual event handlers in CCLScript.

A Flex operator can take any combination of windows and streams as inputs, and produces an output stream or window according to the logic contained in the attached CCLScript scripts.

You can 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.
Related Information

CCLScript [page 17]
CCLScript is a scripting language that brings extensibility to CCL, allowing you to create custom operators and functions that go beyond standard SQL.

6.11.5.1 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. In the Visual editor Palette, in Streams and Windows, select Flex.
2. Click anywhere in the diagram to place the Flex operator.
3. To set the name of the Flex operator, either:
   ○ Click and press F2 to edit the operator name, or,
   ○ In verbose mode, click the edit icon next to the name.
4. Connect the Flex shape to the appropriate input streams or windows.

   Note

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

5. 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.
6. 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, make sure the Flex shape is selected, 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.
7. (Optional) Add an aging policy.
8. (Optional) Click Set Output Keep Policy and set keep policy options.
Related Information

Specifying a Retention Policy [page 66]
The keep policy determines the basis for retaining rows in a window.

Setting an Aging Policy [page 115]
An aging policy can be set to flag records that have not been updated within a defined interval. This is useful for detecting records that may be "stale".

6.11.6 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. 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.

6.11.7 Setting an Aging Policy

An aging policy can be set 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 element.
Procedure

1. Select Set Aging Policy and set values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aging Time</td>
<td>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. When the record is updated (or the Aging Time Field changes), the timer will be reset. The period can be specified in hours, minutes, seconds, milliseconds, or microseconds.</td>
</tr>
<tr>
<td>Aging Field</td>
<td>The field in the record that must be 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. By default, this value is 1.</td>
</tr>
<tr>
<td>(Optional) Max Aging Field Value</td>
<td>The maximum value that the aging field can be incremented to. If not specified, the aging field is incremented once.</td>
</tr>
<tr>
<td>(Optional) Aging Time Field</td>
<td>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.</td>
</tr>
</tbody>
</table>

2. (Optional) Double-click the policy to edit its parameters.

Results

When the project runs, records accumulate until the Aging Time or the Max Aging Field Value is reached. On an update to a record, the age is reset to 0.

6.11.8 Monitoring Streams for Errors

Modify a project to use error streams to keep track of errors in other streams in the project.

Context

Error streams collect information from other streams about errors. Use error streams for debugging projects in development and monitoring projects in a production environment.
Procedure

1. Identify the project and the specific streams to monitor.
2. Determine whether you want to use multiple error streams. Determine the visibility for each error stream.
3. Create the error stream in the project.
4. Display information from the error stream.

6.11.8.1 Creating an Error Stream

Add a special type of stream that collects errors and the records that cause them from other streams in a project.

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. In the Visual editor, open the project.
2. Click the error stream shape in the Palette, then click an empty area in the diagram.
3. Click the + (plus) sign.
   You see a list of streams in the project that can be monitored.
4. 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.
6.11.8.2 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. In the Visual editor, open the project.
2. 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.
3. 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.

6.11.8.3 Modifying an Error Stream

When you are debugging a project in development or monitoring a project in production mode, you may want to change the specific streams that an error stream is monitoring.

Context

Note

An error stream cannot monitor other error streams.

Procedure

1. In the Visual editor, open the project.
2. Locate the Error Stream shape in the work area and review the list of input streams.
3. Click the + (plus) sign, then click each stream to monitor. 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.

4. 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.

6.11.9 Guaranteed Delivery and Persistent Subscribe Pattern

Guaranteed delivery (GD) and persistent subscribe pattern (PSP) are delivery mechanisms that support high availability. They ensure that data continues to be processed from a stream or window even if the SAP ESP server fails, the destination (third-party server) fails, or the destination does not respond for a period of time.

SAP recommends that you use guaranteed delivery rather than persistent subscribe pattern if possible. 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 need to guarantee delivery of data from a delta stream.
- 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.

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 (not on delta streams) and each GD stream or window requires a log store.

You can specify GD support for a window in Studio or in the CCL. (See the SAP Event Stream Processor: CCL Reference for CCL details.) A GD window supports multiple GD subscribers as well as both GD and non-GD subscriptions. To use GD, you must also:

- 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 Event Stream Processor: SDK Guide.

SAP Event Stream Processor projects that have at least one GD-enabled stream or window provide information on GD in two metadata streams:
- _ESP_Streams - tells you whether GD is enabled for the stream or window
- _ESP_GD_Sessions - lists active and inactive GD sessions, identifies the sequence number of the last event committed for each stream or window a GD session subscribes to and the last update time for an entry, and identifies the user associated with this session (that is, the user who initiated the subscription).

For details on these metadata streams, see the SAP Event Stream Processor: Configuration and Administration Guide.

You can subscribe to the metadata streams using the C++ SDK, the Java SDK, or the .NET SDK. You can also monitor the streams yourself using `streamingsubscribe` (see the SAP Event Stream Processor: Utilities Guide) or the Studio Server view. Event Stream Processor stores data from the _ESP_GD_Sessions metadata stream in a special metadata log store so it will be available after a crash.

Consistent recovery mode ensures that if the server restarts, it recovers the state of all streams and windows in a project to the last successful checkpoint state—provided you have followed the rules related to assigning streams and windows to log stores. Consistent recovery is achieved by checkpointing all log stores atomically. If any checkpoints fail (which happens, for example, when the server shuts down in the middle of a checkpoint or there is not enough space in the log stores), Event Stream Processor rolls all the log stores back to the last successful checkpointed state. See the SAP Event Stream Processor: Developer Guide for more information on consistent recovery.

Use the Auto Checkpoint project option to set the number of input transactions that trigger a checkpoint. More frequent checkpoints reduce the risk of data loss; less frequent checkpoints reduce the burden on system resources and may improve performance. Note that the frequency N you set with this option only ensures that a checkpoint happens at least every N transactions. Checkpoints might happen at other times if the system decides that it is necessary or if a publisher issues a commit when the server is running in consistent recovery mode.

**Note**

SAP does not recommend using consistent recovery mode in projects where active-active HA mode is also enabled. Because ESP 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.

**Persistent Subscribe Pattern**

You 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 PSP and GD parameters that are unique to them. For more information, see the *SAP Event Stream Processor: Adapters Guide*.

**Related Information**

*Log Store Guidelines* [page 105]

Special considerations for using log stores.
6.11.9.1 Adding a Persistent Subscribe Pattern

Set up persistent subscribe pattern (PSP) for any element in a project.

**Context**

Create a PSP for a single subscriber. Start with an element such as a stream or window. When you enable PSP on the element, two additional shapes appear in the model: a control window (also known as a truncate window), and a Flex operator with a log store.

i Note

SAP recommends that you use guaranteed delivery instead of PSP whenever possible. See Guaranteed Delivery and Persistent Subscribe Pattern for more information.

**Procedure**

1. Open the project in the Visual editor and choose the element that you would like 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>`. This Flex operator is also known as the log window.
- A truncate window named `<-PSP-shape-name>_truncate<number>`. The truncate window is sometimes called the control window.

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.
6.11.9.2 Adding Guaranteed Delivery

Enable guaranteed delivery (GD) for a stream or window when you need to ensure that the element’s 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 Studio projects.)
  - (Optional) Enable the Auto Checkpoint project deployment option.

Procedure

1. Choose the stream or window you would like 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 drop-down, select the log store you created for this element’s GD logs.
   c. (For GD windows only—skip this step for GD streams.) Select the General tab. In the Store field, enter the name of the log store you created for this GD window.

Related Information

Creating a Log Store [page 102]
Create a log store to allow recovery of data in a window in the event of a server shutdown or failure.
6.11.10 Automatic Partitioning

Automatic partitioning is the creation of parallel instances of an element and splitting input data across these instances. This can improve the performance of an element and complex projects, which perform computationally expensive operations such as aggregation and joins.

You can create parallel instances of a delta stream, stream, window, or module. Reference streams, unions, inputs, adapters, splitters, and error streams cannot use partitioning.

The partitioning degree is the natural number of parallel instances you wish to create for a given element (delta stream, stream, window, or module). As an alternative to specifying the partitioning degree as a constant, you can specify it using an integer parameter with an optional default value. You can then provide the actual value for the parameter in the CCR project configuration file.

The partitioning function is effectively a demultiplexer which determines the target parallel instances for a given partitioning key. There are three valid types of partition functions: ROUNDROBIN, HASH, and CUSTOM. Choose a type based on the calculations you are performing on the input data. For example, ROUNDROBIN is sufficient for stateless operations like simple filters, but not for aggregation as this would produce differing results. HASH is necessary for grouping records together, but grouping may not evenly distribute the data across instances.

CUSTOM Partitions

The CUSTOM partitioning function is defined as an inline function which does not take any parameters. This function creates an implicit global parameter called `<targetName>_partitions` where `<targetName>` represents the name of the current element you are partitioning and `partitions` is a fixed part of the parameter name. For example, if you are partitioning an output window called `maxPriceW`, use `maxPriceW_partitions` as the global parameter name. The value of this parameter is equal to the number of partitions.

The CUSTOM partitioning function returns an integer which determines the parallel instance that should receive a given event (row). A modulo operation applies to this result, which ensures that the returned instance number is greater than or equal to zero and is less than the number of available instances. This prevents runtime errors. For example, if you create three partitions, those partitions will have the IDs 0, 1, and 2.

```plaintext
BY priceW
{
    integer hashValue := ascii(substr(priceW.isin,1,1));
    return hashValue % maxPriceW_partitions;
}
```

Ordering of Partitioned Results

Note that for the same input data the output of partitioned elements may differ from the output of a non-partitioned element. This is caused by the fact that:

- operating systems schedule threads in a non-deterministic way, and
- parallel execution of instances using multiple operating system threads introduces indeterminism, and
- to maximize the throughput of the partitioned element, no explicit synchronization between parallel instances takes place.
The stream partitions which are instantiated by the ESP server at runtime are local and cannot be subscribed or published to. However, these streams are visible in Studio so you can view their utilization and adjust the partition count accordingly.

**Restrictions**

You cannot apply the PARTITION BY clause to these elements: inputs, splitters, unions, reference streams, and adapters. Doing so results in a syntax error. However, you can partition these elements within a module that you are partitioning.

**Example: Roundrobin Partitioning**

Here is an example of ROUNDROBIN partitioning on a CCL query with one input window (TradeWindow):  

```cql
create input window TradeWindow
schema (
  Ts BIGDATETIME,
  Symbol STRING,
  Price MONEY(2),
  Volume INTEGER)
primary key (Ts);
create output window TradeOutWindow
schema (
  Ts BIGDATETIME,
  Symbol STRING,
  Price MONEY(2),
  Volume INTEGER)
primary key (Ts)
PARTITION
  by TradeWindow ROUNDROBIN
PARTITIONS 2
as
  SELECT * FROM TradeWindow
  WHERE TradeWindow.Volume > 10000;
```

This example partitions the output window, TradeOutWindow, using ROUNDROBIN partitioning and creates two parallel instances.

**Example: HASH Partitioning**

Here is an example of HASH partitioning on a CCL query with one input window (priceW):  

```cql
create input stream priceW
schema (isin string, price money(2));
create output window maxPriceW
schema (isin string, maxPrice money(2))
primary key deduced keep 5 minutes
PARTITION
```
This example partitions the output window, maxPriceW, using HASH partitioning and creates five parallel instances.

Here is an example of HASH partitioning on one of the input windows (priceW) on a join while the other input window (volumeW) is broadcast:

```sql
create input window priceW
schema (isin string, price float) primary key (isin) keep 5 minutes;
create input window volumeW
schema (isin string, volume integer) primary key (isin) keep 5 minutes;
create output window vwapW
primary key deduced keep 1 minute
PARTITION
by priceW HASH (isin) PARTITIONS 2
as
SELECT priceW.isin, vwap(priceW.price, volumeW.volume) vwap_val
FROM priceW LEFT JOIN volumeW ON priceW.isin = volumeW.isin
group by priceW.isin;
```

This example partitions the output window, vwapW, using HASH partitioning and creates two parallel instances.

Example: CUSTOM Partitioning

Here is an example of CUSTOM partitioning on a CCL query with two input windows (priceW and volumeW):

```sql
create input window priceW
schema (isin string, price float)
primary key (isin) keep 5 minutes;
create input window volumeW
schema (isin string, volume integer)
primary key (isin) keep 5 minutes;
create output window vwapW
schema (isin string, vwap float)
primary key deduced
partition
by priceW {
  return ascii(substr(priceW.isin,1,1)) % vwapW_partitions;
},
by volumeW {
  return ascii(substr(volumeW.isin,1,1)) % vwapW_partitions;
}
partitions 2
as
SELECT priceW.isin, vwap(priceW.price, volumeW.volume) vwap_val
FROM priceW LEFT JOIN volumeW ON priceW.isin = volumeW.isin
group by priceW.isin;
```
This example partitions the output window, \texttt{vwapW}, using a CUSTOM partitioning function and creates two parallel instances.

### 6.11.10.1 Creating a Partition

Partition an existing delta stream, stream, window, or module in the Visual editor.

#### Prerequisites

(Optional) To use a parameter to specify the number of partitions you wish to create, create a global parameter first. See *Declaring a Parameter* for detailed steps on creating a parameter.

#### Procedure

1. Click \textit{Add Partition Policy}.
2. Select a partition policy.
3. In the Properties view, specify a value or parameter for the Number of Partitions to create. To use a parameter, either enter the parameter name or click \textit{Select} and select the parameter from the list.
4. Select the new partition and edit its properties in the Properties view:

<table>
<thead>
<tr>
<th>Partition Policy</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ROUNDROBIN}</td>
<td>1. Select a source for the partition.</td>
</tr>
</tbody>
</table>
| \texttt{HASH} | 1. Select a source for the partition.  
             2. Select the columns to use for the HASH function and click \textit{Add}. |
## Partition Policy

<table>
<thead>
<tr>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select a source for the partition.</td>
</tr>
<tr>
<td>2. On the Expression tab, edit the partition function. To show syntax completion proposals, press Ctrl+Space.</td>
</tr>
<tr>
<td>3. To ensure that the function returns a valid partition ID, apply a modulo operation in the return statement and use the implicit global parameter of the partition as its divisor. For example, for an output window called maxPriceW, use maxPriceW_partitions as the divisor:</td>
</tr>
</tbody>
</table>

```ccl
BY priceW |
    integer hashValue :=
    ascii(substr(priceW.isin,1,1)); |
    return hashValue %
maxPriceW_partitions;
``` |

### Related Information

**Declaring a Parameter** [page 128]

Create a new (global) parameter using Studio.

### 6.11.11 Declaring a Parameter

Create a new (global) parameter using Studio.

### Context

Parameters are constants that you set during project setup. Parameters can use only basic datatypes, and must be 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. See the Advanced CCL Programming Techniques section of the SAP Event Stream Processor: Developer Guide for more information.
From the SAP ESP Authoring perspective:

**Procedure**

1. In the Outline view, right-click Statements or one of its child folders (those directly below it) and select Modify > Edit Global Declaration(s).
2. 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.

**Next Steps**

Edit parameter values using the project configuration file (CCR) editor. See Setting Parameters in Project Configuration in the SAP Event Stream Processor: Configuration and Administration Guide for more information.

**Related Information**

*Creating a Partition* [page 127]

Partition an existing delta stream, stream, window, or module in the Visual editor.

**6.11.12 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. In the Visual editor workspace, in the Palette menu under the Streams and Windows category, select Splitter.
2. Select a location in the diagram and click to add the shape.
3. 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.
4. (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.
5. Connect the splitter to a single Input Stream or a Window.
6. (Optional) Add or remove Column Expressions for the splitter.
7. Create the splitter logic using Add When and Add Else. This will create the splitter output elements.
8. (Optional) Connect the splitter output elements of the splitter to other Streams or Windows.

6.11.13 Adding a Binding to a Project

Add a binding into your project in several ways using the Visual editor in the SAP ESP Authoring perspective.

Prerequisites

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

Context

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

Procedure

1. Add a binding by dragging and dropping a stream or window from the Project Explorer:
   a. Open the project to which you wish to add a binding in the Visual editor.
   b. In your Project Explorer, expand the folder of the second project you wish to bind to the first project.
   c. Click the output stream/window you wish to bind to the first project and drag and drop it into the first project’s diagram.
      - A new input stream/window is automatically created and added to the diagram of the first project.
      - The new stream/window is named using the format <FirstProjectName>_<StreamOrWindowName> and has an identical inline schema to the schema of the stream/window from the second project.
1. Add a binding from a stream or window shape within a diagram:
   a. Open the project to which you wish to add a binding in the Visual editor.
   b. In the project diagram, click the binding icon in the stream/window shape that you wish to bind to another stream/window.
      A new Select Binding Target window opens.
   c. Select the stream/window that you wish to bind to the first stream/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 window/stream 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.
      ○ Use the Project Configuration editor to specify a value for the Cluster property, and optionally, values for other properties.

2. Add a binding from a stream or window shape within a diagram:
   a. Open the project to which you wish to add a binding in the Visual editor.
   b. In the project diagram, click the binding icon in the stream/window shape that you wish to bind to another stream/window.
      A new Select Binding Target window opens.
   c. Select the stream/window that you wish to bind to the first stream/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 window/stream 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.
      ○ Use the Project Configuration editor to specify a value for the Cluster property, and optionally, values for other properties.

3. Add a binding by clicking within a diagram:
   a. Open the project to which you wish to add the binding in the Visual editor.
   b. Right-click any blank space in the diagram, and select Add Binding (in CCR).
      A new Binding (in CCR) window opens.
   c. Select a stream/window for the Local stream/window field and a remote stream/window for the Remote stream field.
   d. Click OK.
      ○ A new tab for the Project Configuration editor opens within the Visual editor.
      ○ An icon appears to the left side of the window/stream 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.
      ○ Use the Project Configuration editor to specify a value for the Cluster property, and optionally, values for other properties.

4. Save your changes to the project, as well as the project configuration file.
6.11.14 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 a primary key, either explicit or deduced. When connecting a stream to a derived window, you assign a primary key one of two ways: using 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, it is recommended that only additive functions, such as `valueInserted()`, `sum()`, `avg()`, and `count()`, are used 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. See Improving Aggregation Performance in the SAP Event Stream Processor: Developer Guide for more information. 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.

6.11.14.1 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 connected to a stream.

Procedure

1. Open a new or existing ESP 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.
6.11.14.2 Connecting a Stream to a Derived Window Using nextval()

Use the `nextval()` function to set a primary key for a derived window connected to a stream.

**Procedure**

1. Open a new or existing ESP 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 drop down menu for the derived window; select *Modify* and then *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.
CCL Editor Authoring

The CCL editor is a text authoring environment within SAP Event Stream Processor Studio for editing CCL code. 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.

Most users new to Event Stream Processor find it easier to get started in the Visual editor. As you gain experience with the product, and learn to successfully compile and run a simple project, you may want to use the CCL editor to add 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
- User-defined functions
- Reusable modules and schemas that can be used multiple times in a project, or across projects

The ESP server only accepts user-defined functions if the enable-udfs property is set to true in the cluster configuration. By default, this property is set to false and projects containing external user defined functions (UDFs) cannot run successfully. See Enabling External User Defined Functions in the SAP Event Stream Processor: Cockpit Guide for detailed instructions.

For CCL language details, see the SAP Event Stream Processor: CCL Reference.

7.1 Editing in the CCL Editor

Update and edit CCL code as text.

Procedure

1. Select the SAP ESP Authoring perspective.
2. In Project Explorer, expand the project container, and double-click the .ccl file name to open it in the CCL editor.
3. Begin editing text in the CCL editor window.

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.
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. Any Windows directory paths must therefore be specified 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. Choose File > Save (Ctrl+S) to save the .ccl file and the project.

Related Information

Project Explorer [page 29]
Organize and navigate among your projects using the Project Explorer, which provides a tree-structured hierarchy of folders and files.

Switching Between the CCL and Visual Editors [page 56]
Change between the two editors to maximize Studio’s flexibility for creating and editing a project.

Compiling a Project [page 147]
Produce an executable .ccx file from CCL code. CCL code must be compiled to produce an executable to run on SAP Event Stream Processor.

7.2 CCL Editor Features

Several features simplify the process of editing CCL code in the Studio CCL editor.

Table 10: CCL Editor Features

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

SAP Event Stream Processor: Studio Users Guide
CCL Editor Authoring
© 2014 SAP SE or an SAP affiliate company. All rights reserved. 135
7.3 Keyboard Shortcuts in the CCL Editor

Use keyboard shortcuts to access various functions quickly within the CCL editor.

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

To add a keyboard shortcut:
1. Select Window ➤ Preferences.
2. Select General ➤ Keys.
3. Select a command from the list.
4. In the Binding field, add the new shortcut.
5. Click OK.

7.4 Searching for Text

Find text in CCL code.

Procedure

1. Choose Search ➤ File.
   You can also start a new search from the link in the Search view, when no search results are visible.
2. Enter search criteria in the dialog.
3. Choose either:
   - Search to show results, or
   - Replace to replace results.
4. Review results in the Search view and choose from options in the Search toolbar.

Tip

Double-click a match to highlight it in the CCL editor.
7.5 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`, `CREATE WINDOW` and `CREATE DELTA STREAM` 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, see the SAP Event Stream Processor: Developer Guide:

- In Statements, see `CREATE STREAM`, `CREATE WINDOW`, and `CREATE DELTA STREAM` statements for clauses they support
- In Clauses, see syntax and usage details

Related Information

Simple Queries [page 69]
Accomplish most common querying tasks using a set of queries available in the Visual editor: filter, aggregate, join, compute, union, and pattern.

7.6 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

1. 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 }
   ```
7.7 CCL Functions

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

SAP Event Stream Processor supports:

- Built-in functions - including aggregate, scalar and other functions
- User-defined CCLScript functions
- User-defined external C/C++ functions
- User-defined external Java functions

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

The ESP Server only accepts user-defined functions if the enable-udfs property is set to true in the cluster configuration. By default, this property is set to false and projects containing external user defined functions (UDFs) cannot run successfully. See Enabling External User Defined Functions in the SAP Event Stream Processor: Cockpit Guide for detailed instructions.

Note

Because Java and C/C++ user-defined functions are not run in a secure sandbox, it is possible to write functions that access the file system or network I/O. Therefore it is important to do a thorough code review with a focus on security before deploying a user-defined function to ESP.

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.
7.8 Operators

CCL supports a variety of numeric, nonnumeric, and logical operator types.

Arithmetic Operators

Arithmetic operators are used to negate, add, subtract, multiply, or divide numeric values. They can be applied to numeric types, but they also support mixed numeric types. Arithmetic operators can have one or two arguments. A unary arithmetic operator returns the same datatype as its argument. A binary arithmetic operator chooses the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data-type, and returns that type.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>3+4</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>7-3</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>3*4</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>8/2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (Remainder)</td>
<td>8%3</td>
</tr>
<tr>
<td>^</td>
<td>Exponent</td>
<td>4^3</td>
</tr>
<tr>
<td>-</td>
<td>Change signs</td>
<td>-3</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
<td>++a (preincrement)</td>
</tr>
<tr>
<td></td>
<td>Preincrement (++&lt;argument&gt;) value is incremented before it is passed as an argument</td>
<td>a++ (postincrement)</td>
</tr>
<tr>
<td></td>
<td>Postincrement (&lt;argument&gt;++) value is passed and then incremented</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
<td>--a (predecrement)</td>
</tr>
<tr>
<td></td>
<td>Predecrement (--&lt;argument&gt;) value is decremented before it is passed as an argument</td>
<td>a-- (postdecrement)</td>
</tr>
<tr>
<td></td>
<td>Postdecrement (&lt;argument&gt;--) value is passed and then decremented</td>
<td></td>
</tr>
</tbody>
</table>

Comparison Operators

Comparison operators compare one expression to another. The result of such a comparison can be TRUE, FALSE, or NULL.

Comparison operators use this syntax:

```
expression1 comparison_operator expression2
```
### Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equality</td>
<td>a0=a1</td>
</tr>
<tr>
<td>!=</td>
<td>Inequality</td>
<td>a0!=a1</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Inequality</td>
<td>a0&lt;&gt;a1</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>a0&gt;a1</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>a0&gt;=a1</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>a0&lt;a1</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>a0&lt;=a1</td>
</tr>
<tr>
<td>IN</td>
<td>Member of a list of values. If the value is in the expression list’s values, then the result is TRUE.</td>
<td>a0 IN (a1, a2, a3)</td>
</tr>
</tbody>
</table>

### Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Returns TRUE if all expressions are TRUE, and FALSE otherwise.</td>
<td>(a &lt; 10) AND (b &gt; 12)</td>
</tr>
<tr>
<td>NOT</td>
<td>Returns TRUE if all expressions are FALSE, and TRUE otherwise.</td>
<td>NOT (a = 5)</td>
</tr>
<tr>
<td>OR</td>
<td>Returns TRUE if any of the expressions are TRUE, and FALSE otherwise.</td>
<td>(b = 8) OR (b = 6)</td>
</tr>
<tr>
<td>XOR</td>
<td>Returns TRUE if one expression is TRUE and the other is FALSE. Returns FALSE if both expressions are TRUE or both are FALSE.</td>
<td>(b = 8) XOR (a &gt; 14)</td>
</tr>
</tbody>
</table>

### String Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenates strings and returns another string.</td>
<td>'go' + 'cart'</td>
</tr>
</tbody>
</table>

**Note**

The + operator does not support mixed datatypes (such as an integer and a string).

### LIKE Operator

May be used in column expressions and WHERE clause expressions. Use the LIKE operator to match string expressions to strings that closely resemble each other but do not exactly match.
### Operator

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax and Meaning</th>
<th>Example Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIKE</td>
<td>Matches WHERE clause string expressions to strings that closely resemble each other but do not exactly match.</td>
<td>Trades.StockName LIKE &quot;%Corp%&quot;</td>
</tr>
<tr>
<td></td>
<td><code>compare_expression LIKE pattern_match_expression</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The LIKE operator returns a value of TRUE if <code>compare_expression</code> matches <code>pattern_match_expression</code>, or FALSE if it does not. The expressions can contain wildcards, where the percent sign (%) matches any length string, and the underscore (_) matches any single character.</td>
<td></td>
</tr>
</tbody>
</table>

### [] Operator

The [] operator can be used with dictionaries and vectors, and to look up a window record by key field.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax and Meaning</th>
<th>Example Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>In the case of dictionaries, <code>dictionary_name[key_value]</code> references the value held in that dictionary location.</td>
<td><code>rec := mydictionary[abc];</code></td>
</tr>
<tr>
<td>[]</td>
<td>For vectors, <code>vector_name[index]</code> gets the value from a specific index position in a vector.</td>
<td><code>rec := myvector[2]</code></td>
</tr>
<tr>
<td>[]</td>
<td>With an input window, you can use [] to lookup a record in the window with a key field.</td>
<td><code>Trades_stream[ [TradeId = 10;]] ]</code> would get a record in Trades with the TradeId key field having a value of 10. <code>Trades_stream[AnotherEvent]</code> would get a record called Another Event from another window which has all the key columns of the Trades_stream, with the same key column name.</td>
</tr>
</tbody>
</table>
Order of Evaluation for Operators

When evaluating an expression with multiple operators, the engine evaluates operators with higher precedence before those with lower precedence. Those with equal precedence are evaluated from left to right within an expression. You can use parentheses to override operator precedence, since the engine evaluates expressions inside parentheses before evaluating those outside.

Note
The ^ operator is right-associative. Thus, a ^ b ^ c = a ^ (b ^ c), not (a ^ b) ^ c.

The operators in order of preference are as follows. Operators on the same line have the same precedence:

- +,- (as unary operators)
- ^
- *., /, %
- +,- (as binary operators and for concatenation)
- =, !=, <>, <, >, <=, >= (comparison operators)
- LIKE, IN, IS NULL, IS NOT NULL
- NOT
- AND
- OR, XOR

7.9 Adding Tooltip Comments for the Visual Editor in CCL

Write comments in CCL that appear as tooltips for shapes in the Visual editor.

If you want comments to appear as tooltips in the Visual editor, you must insert a comment immediately preceding the declaration statement for the corresponding shape in this form:

```ccl
/**InsertTooltipCommentHere*/
```

Here is an example, in CCL, of a tooltip comment for an Input Window shape in the Visual editor.

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

Comments inputted into the CCL editor in this manner will appear as tooltips in the Visual editor when the corresponding shapes are hovered over.

Note
'Show comments in tooltip' must be enabled in Preferences.
8 Project Configuration Files

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 ESP Authoring perspective.

Configuration files maintain all run-time properties outside the CCL. Thus, you can maintain CCL and CCX files under version control, while varying run-time properties. This allows a project to be moved from a test environment to a production environment without modifying the CCL and CCX files.

By default, when a new project is created, a new project configuration file is also created. One project may have multiple configuration files attached to it, so you can manually create new project configurations. However, opening your project using SAP Event Stream Processor Studio will only deploy the original CCR file created with the project. This CCR file will have 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 Event Stream Processor: Utilities Guide.

See Project Configurations in SAP Event Stream Processor: Configuration and Administration Guide for more information on editing CCR file parameters.

8.1 Advanced Project Deployment Options

Project deployment options determine how your project is deployed in a cluster and how it functions at runtime. Set these parameters, including project options, active-active instances, failover intervals, and project deployment type options, within Studio.

Active-Active Deployments

To deploy a project in active-active or HA (high availability) mode, set <Project ha="true"/> in the CCR file. In an active-active deployment, two instances of a project run simultaneously in a cluster. Active-active projects are typically configured so that the cluster starts the two instances of the project on different nodes (hosts). This feature avoids the risk of a single point of failure at the project level.

One instance of the project is elected as the primary instance. If one of the instances is already active, it is the primary instance. If the failed instance restarts, it assumes the secondary position and maintains this position unless the current instance fails or is stopped.
Project Options

Project options are used as runtime parameters for the project, and include a predefined list of available option names that reflect most command line entries.

Instances

When a project is deployed in HA (active-active) mode, two instances are created: primary and secondary. Whether the project is in HA mode or not, you can set affinity and cold failover options for each instance, including failover intervals and failure per interval options. Non-HA projects have one instance, numbered 0 (zero). HA project instances are numbered 0 and 1. Some commands require instance numbers to identify instances of a project.

Failover

A project fails when it does not run properly or stops running properly. If cold failover is enabled, a failover occurs when a failed project switches to another server to continue processing. Failover typically results in a project restart, though a strong positive affinity to a node that is not available can prevent a project from restarting. Restarts can be limited based on failure intervals and restarts per interval. Failover options, accessed using an instance configuration, include:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failover</td>
<td>Either enabled or disabled. When disabled, project failover restarts are not permitted. When enabled, failure interval and failures per interval fields can be accessed and restarts are permitted.</td>
</tr>
<tr>
<td>Failures per interval</td>
<td>Specifies the number of restarts the project can attempt within a given interval. This count resets to zero if you restart the project manually or if failures are dropped from the list because they are older than the size of the interval.</td>
</tr>
<tr>
<td>Failure interval</td>
<td>(Optional) This specifies the time, in seconds, that makes up an interval. If left blank, the interval time is infinite.</td>
</tr>
</tbody>
</table>

Affinities

Affinities limit where a project runs or does not run in a cluster. There are two types of affinities:

- **Controller** – for active-active and non-active-active configurations. Controller affinities let you establish rules and preferences as to which controller nodes your project can run on. A project can have affinities for more than one node, but it can have a strong positive affinity for only one node.
- **Instance** – only for active-active configurations. The two instances of an active-active project can have affinities for each other. For example, if you want such instances never to run on the same node, set strong negative instance affinities. If you want them to avoid running on the same node if possible, set weak negative instance affinities.
Define these parameters for each affinity:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the object of the affinity, that is, the controller name or instance name that the affinity is set for. For instance affinities, the affinity for one instance must refer to the second instance.</td>
</tr>
<tr>
<td>Strength</td>
<td>Specify <strong>Strong</strong> or <strong>weak</strong>. Strong requires the project to run on a specific controller, and no others. If you have strong positive affinity set for a controller node, and that node fails, the failover process tries to restart the project on that node. If the node has not recovered, the project restart fails and you must restart manually. A weak positive affinity causes the project starts on the preferred controller if possible, but if that controller is unavailable, it may start on another available controller.</td>
</tr>
<tr>
<td>Charge</td>
<td>Specify <strong>Positive</strong> or <strong>negative</strong>. If positive, the project runs (for a strong affinity) or prefers to run (for a weak affinity) on the named controller. If negative, the project does not run (or prefers not to run) on the named controller.</td>
</tr>
</tbody>
</table>

**Processor Affinity**

Processor Affinity lets you set a project to only run on specified processors. In HA (active-active) mode, processor affinities can be set separately for each instance.
9 Using the Data Services View

The ESP server accesses external databases by using data service definitions which are managed and stored in separate databases for each cluster. Use the Data Services view to add, edit or delete database service definitions for local and remote clusters.

You can also migrate database services defined in the service.xml file of previous releases of ESP using the service.xml migration command of the streamingclusterutil command line utility.

Click the SAP ESP Authoring tab at the top of the Studio main window to see the Data Services View. If the Data Services view is not visible, from the main menu select Window Show View Other SAP Event Stream Processor Data Services. The Data Services view is automatically populated with the connections you create to remote servers in the Server view, which is visible in the SAP ESP Run-Test perspective.

Adapters that require database access obtain connections from the database manager by specifying the service that the connection is created for. For example, you can define services for connecting to an SAP ASE database through JDBC and to SQL Server through ODBC. At runtime, the adapter obtains a connection from the database manager based on the properties in the data service, and executes queries over it.

10 Running and Testing Projects

Test a project by compiling and running it on a server, accessing and filtering streams, saving and uploading data to the SAP Event Stream Processor server, and setting project configurations.

10.1 Starting the Run-Test Perspective

Access the SAP ESP Run-Test perspective for toolbars and views that simplify testing, monitoring, debugging, and examining Event Stream Processor projects.

Procedure

1. Click the SAP ESP Run-Test tab at the top of the Studio main window to see the SAP ESP Run-Test perspective.
   If the tab is not visible, from the main menu select Window > Open Perspective > Other > SAP ESP Run-Test.

10.2 Compiling a Project

Produce an executable .ccx file from CCL code. CCL code must be compiled to produce an executable to run on SAP Event Stream Processor.

Procedure

1. (Optional) Set CCL compiler options.
   a. Choose Window > Preferences.
   b. Click SAP Event Stream Processor > Run Test and go to Compiler output directory.
   c. To change the directory for your compiled projects, click Change..., select a directory, and click OK.
   d. 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.

2. In the SAP ESP Authoring perspective, in Project Explorer, expand the tree view to show the .ccl file for the project.
3. Select and open the `.ccl` project that you want to compile.

4. 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.

5. To compile and run the project: click `Run ESP Project`.

   The project automatically compiles and runs. The Server View in the SAP ESP 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**

   When you have Studio installed on a separate machine from the SAP ESP server, you cannot compile projects on the default local cluster. Instead, connect to the SAP ESP server and then compile your project. See Connecting to a Remote Cluster [page 20] for detailed instructions.

**Results**

Studio silently 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. And, 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.

**10.2.1 Viewing Problems**

Use the Problems view to view error details when trying to validate, upload, and compile projects.

**Prerequisites**

Open the SAP ESP Authoring perspective.
Procedure

1. Click on a problem in Problems view, or expand the group to see individual errors.
   By default, Problems view is at the bottom of the screen, and problems are grouped by severity.
   Error details appear in 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.

2. 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.

3. Right-click an item to choose from the context menu:

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to</td>
<td>Highlight the problem in the .ccl file. The CCL editor opens in read-only mode.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy error details to the clipboard. When you exit Studio, the contents of problems view are removed. Use this option to save off errors.</td>
</tr>
<tr>
<td>Show in</td>
<td>Display details in Properties view.</td>
</tr>
<tr>
<td>Quick Fix</td>
<td>(Disabled)</td>
</tr>
<tr>
<td>Properties</td>
<td>Display details in a dialog box.</td>
</tr>
</tbody>
</table>

4. (Optional) Click the View menu dropdown to see more options.

5. Click the **Console** tab to view compiler results.

10.3 Caching Project Settings

Context

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

Items cached include:

- List of Stream View stream tabs
- Current Manual Input Stream
- Current SQL Query, Playback, Upload, Monitor, Event Tracer projects
To set this preference:

Procedure

1. Click *Window > Preferences > SAP Event Stream Processor > Run Test.*
2. Select *Always start previously running run-test views on project start.*
3. Uncheck *Clear all cached run-test views.*

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.

10.4 Running a Project

Running a project automatically starts the project either on a default local cluster or on another connected cluster, enabling you to upload data from a file, manually enter data, and view streams in that project.

Prerequisites

To run a project, ensure that one or more connected workspaces are available.

When you have Studio installed on a separate machine from the SAP ESP server, you cannot run projects on the default local cluster. Instead, connect to the SAP ESP server and then run your project. See *Connecting to a Remote Cluster* [page 20] 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 ESP Authoring or the SAP ESP Run-Test perspective) to run the project in the default/most recently used workspace, or,
   - Click the drop-down 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 ESP 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. To redeploy, remove
the project from the SAP ESP Run-Test server view, and then restart the project from the SAP ESP Authoring perspective.

10.4.1 Server View

The Server View shows SAP Event Stream Processor servers available for connecting and running projects. You can:

- Connect a project, enabling a local or remote cluster
- 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
- Filter metadata streams (default)

Metadata streams are created automatically, and are typically used by administrators in a production system to obtain health and performance information about the currently running project. For details of what each stream contains, see Metadata Streams in the SAP Event Stream Processor: Configuration and Administration Guide.

Related Information

- Cluster Connectivity in Studio [page 18]
  In Studio, you can run a project on either a local or a remote cluster, using any of three methods of authentication. You can automatically run a project on a local cluster without needing to perform any prior steps. To run a project on a remote cluster, however, first connect to that cluster. This requires some simple configuration steps.

- Performance Monitor [page 155]
  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.

- Event Tracer View [page 162]
  The Event Tracer is one of the tools used to debug data flow. It shows the impact an event has on each stream and window of the project.
10.4.2 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 ESP 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**.

10.4.3 Specifying 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. For example, a stream containing three ticker symbols 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.

Context

There are two preferences that control the subscription feature in Studio: **StreamViewer pulsed subscribe interval** and **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 that 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.
To change the default settings:

**Procedure**

1. Choose Window ➤ Preferences.
2. In the left pane, expand SAP Event Stream Processor ➤ 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.

### 10.4.4 Uploading Data to the Server

Use the File Upload tool to load event data from files into a running project. Normally used in testing a project. 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 ESP 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.

   **Note**
   
   The ESP server supports ESP binary (.bin), ESP XML (.xml), and comma-separated values and text (.csv or .txt) files. Regardless of file type, each record in the file must start with the input stream or window name in the first field, followed by the opcode in the second field, followed by the actual contents of the record in the remaining fields.
6. Click **Upload**. A progress bar tracks the upload status.

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

<table>
<thead>
<tr>
<th>UI control</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Remove File</em></td>
<td>Discard a previously selected file from the Input File(s) menu.</td>
</tr>
<tr>
<td><em>Cancel</em></td>
<td>Cancel a file upload currently in progress.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>Any data sent before the upload is cancelled is still processed.</td>
</tr>
<tr>
<td><em>Use Transaction</em></td>
<td>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.</td>
</tr>
<tr>
<td><em>Record Buffer</em></td>
<td>Specify the number of records to group together and process in a single transaction.</td>
</tr>
</tbody>
</table>

### 10.4.5 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 ESP 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.
10.5 Performance Monitor

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 also 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% CPU use, based on a single CPU core. With multicore or multiprocessor environments, a fully black node may be greater than 100%.

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

10.5.1 Running the Monitor

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

Prerequisites

The project must be running before starting the monitor. You can specify a delay by changing the performance timer interval.
Procedure

1. In the SAP ESP Run-Test perspective, select the Monitor view.
2. Click Select Running Project.
3. Click OK.
4. 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 >=.

5. Click Zoom In or Zoom Out to see a larger or smaller view of the diagram.

Related Information

Saving a Performance Diagram as an Image [page 156]

Save a performance diagram.

10.5.2 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 ESP Run-Test perspective. The diagram appears in the Event Tracer window, and can be saved only from that window.

Procedure

1. In the SAP ESP 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.
10.6 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 ESP 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.

10.7 Playback View

The Playback view records in-flowing data to a playback file, and plays the captured data back into a running Event Stream Processor 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.

Table 11: Playback View Options

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select playback file</td>
<td>Select the file format to use with the Event Stream Processor recorder.</td>
</tr>
<tr>
<td>Start playback</td>
<td>Starts playing the current playback file.</td>
</tr>
<tr>
<td>Stop playback</td>
<td>Stops playback or record, closes the associated file and closes the associated playback or record context.</td>
</tr>
<tr>
<td>Start Recording</td>
<td>Prompts the user to select the file in which to store recorded data and starts the Event Stream Processor recorder.</td>
</tr>
<tr>
<td>At Timestamp rate</td>
<td>This slider is used during playback to vary the rate of playback</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>XML/CSV datemask</strong></td>
<td>Applies a datetime mask to read data from the source file. The default datetime format is UTC, or <code>%Y-%m-%dT%H:%M:%S</code>. This option cannot change delimiters, which are the characters that separate each value: <code>&quot;-&quot;, &quot;T&quot;: &quot;,&quot;</code>).</td>
</tr>
</tbody>
</table>

**Table 12: Playback Mode Options**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full rate</strong></td>
<td><em>Full rate</em> indicates that the speed of playback is not imposed by the Studio. <em>Full rate</em> is dependent on factors such as the computer that is running Studio, or network latency.</td>
</tr>
</tbody>
</table>
| **Timestamp column** | The *timestamp column* option tells the recorded file to play back using the timing rate information from a specified column. You must complete the *timestamp column* to use it. During playback, timestamps determine the time interval between records. When using the *timestamp column*, you can select *Use Recorded Time* to change the project runtime to the time the data was recorded. Select *timestamp column*, check *Use Recorded time*, and enter a column name in the *timestamp Column* field. To return to current time, restart the project. The following datetime datatypes are supported in the *timestamp column* field:  
  - `BIGDATETIME`  
  - `SECONDDATE`  
  - `TIME`  
  - `INTERVAL`  
  - `INTEGER`  
  - `LONG`  
When entering a datemask in the **XML/CSV datemask** field, the following three 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.

The following three datatypes do not support a datemask:  
  - `INTERVAL`  
  - `INTEGER`  
  - `LONG`
The Studio recorder supports these file formats:

- `.xml` (ESP XML)
- `.csv` (comma-separated values)
- `.bin` (ESP Binary)
- `.rec` (Studio recorded file)

Regardless of file type, each record in the file must start with the input stream or window name in the first field, followed by the opcode in the second field, followed by 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`

CSV and XML files have additional formatting rules:

For CSV files, the field must be in an order corresponding 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 allowed provided that the correct number of fields are represented and delimited by commas (or whatever delimiter is used). 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 unless they are first converted to the new binary format using `streamingconvert`. See `streamingconvert` in the SAP Event Stream Processor: Utilities Guide for information about how to convert binary files.

SAP Event Stream Processor 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 ESP server running in a little-endian machine, and vice-versa.
10.7.1 Recording Incoming Data in a Playback File

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

Prerequisites

Connect to an SAP Event Stream Processor server. Have your streams and windows visible in the Stream View.

Procedure

1. Open the SAP ESP 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.

Related Information

Playing Recorded Data [page 161]

View and play previously recorded data in a running Event Stream Processor instance.
10.7.2 Playing Recorded Data

View and play previously recorded data in a running Event Stream Processor instance.

Context

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 playback a file registered in the history. Double-click a history entry to activate it for playback.

   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.
   The data appears in the Stream View, by default, at the rate it was recorded.

Related Information

**Recording Incoming Data in a Playback File** [page 160]

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

10.8 Debugging

The SAP ESP Run-Test perspective contains two tools for debugging data flow and assisting 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.
The debugging tools are for use during project development, not while Event Stream Processor is in production mode. Debugging features are normally disabled. The system must be in Trace mode before you can use 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 Event Stream Processor: Utilities Guide.

## 10.8.1 Event Tracer View

The Event Tracer is one of the tools used to debug 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.

### Table 13: Event Tracer View

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

### Related Information

[Debugging with Breakpoints and Watch Variables](page 164)

Studio allows you to control a running project by enabling tracing, pausing, resuming, and stepping of data flow through Event Stream Processor streams. You can also create breakpoints and watch variables on a running application.
10.8.1.1 Tracing Data Flow in the Event Tracer

Run the Event Tracer from the Event Tracer tab or the Server view.

Prerequisites

Ensure that the ESP server is running.

Procedure

1. In the SAP ESP Run-Test perspective:

<table>
<thead>
<tr>
<th>Method</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Tracer</td>
<td>1. Click the Event Tracer view.</td>
</tr>
<tr>
<td></td>
<td>2. Click <em>Select Running Project</em> to show running projects that contain streams or windows.</td>
</tr>
<tr>
<td></td>
<td>3. Select a running project for the Event Tracer.</td>
</tr>
<tr>
<td></td>
<td>4. Click OK.</td>
</tr>
<tr>
<td>Server View</td>
<td>1. Select the Server View.</td>
</tr>
<tr>
<td></td>
<td>2. To refresh the Server View, click <em>Reconnect All</em>.</td>
</tr>
<tr>
<td></td>
<td>3. Select a running project that contains streams.</td>
</tr>
<tr>
<td></td>
<td>4. Right-click the project node, and select <em>Show in Event Tracer View</em>.</td>
</tr>
</tbody>
</table>

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. To load test data to view the impact on each stream in the Event Tracer tab, either:
   - Click the *Upload File* tab in the toolbar below the Activate Project pane to upload data from a file, or,
   - 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.
10.8.1.2 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 ESP 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.

10.8.2 Debugging with Breakpoints and Watch Variables

Studio allows you to control a running project by enabling tracing, pausing, resuming, and stepping of data flow through Event Stream Processor streams. You can also create breakpoints and watch variables on a running application.

Breakpoints are locations in stream or window input or outputs that stop the flow of data in the Event Stream Processor model. A watch variable inspects the data.

Table 14: Studio Breakpoint Buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace On</td>
<td>Instructs Event Stream Processor to begin tracing (debugging). This parameter must be set to use the Event Stream Processor breakpoint APIs.</td>
</tr>
<tr>
<td>Trace Off</td>
<td>Stops tracing (debugging).</td>
</tr>
<tr>
<td>Step Project</td>
<td>Steps the running Event Stream Processor.</td>
</tr>
<tr>
<td>Pause Project</td>
<td>Pauses playback for projects recorded as .rec files; will not pause other file types.</td>
</tr>
</tbody>
</table>

| Note | When the project is paused, the records from Manual Input and File Upload cannot be updated to streams until the project is resumed. |

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable All Breakpoints</td>
<td>Enables all breakpoints in the list.</td>
</tr>
<tr>
<td>Disable All Breakpoints</td>
<td>Disables all breakpoints in the list.</td>
</tr>
<tr>
<td>Insert Breakpoint</td>
<td>Inserts a breakpoint item into the watch table.</td>
</tr>
<tr>
<td>Insert Watch</td>
<td>Inserts a watch item into the watch table.</td>
</tr>
<tr>
<td>Print Breakpoint Data to Console</td>
<td>Prints the breakpoint and pause state data for the current Event Stream Processor to the console.</td>
</tr>
</tbody>
</table>
The following breakpoint commands initiate long-running operations. Each of these can be cancelled before completion by clicking Cancel Current Step.

Table 15: Breakpoint Commands

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

i Note

Breakpoints and watch variables are persisted to the workspace.

**Related Information**

**Event Tracer View** [page 162]

The Event Tracer is one of the tools used to debug data flow. It shows the impact an event has on each stream and window of the project.

**10.8.2.1 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 an event flows through the breakpoint. The counter is then reset to zero.

**Related Information**

**Adding Breakpoints** [page 166]

Add breakpoints to Event Stream Processor.

**Watch Variables** [page 167]
You can insert watch variables into the watch table of the Breakpoints view in the Debugger to inspect data as it flows through the project.

Adding Watch Variables [page 168]
Add a watch element to a breakpoint.

Pausing SAP Event Stream Processor [page 169]
Pause SAP Event Stream Processor while playing back projects with .rec file types.

Stepping through a Project [page 170]
Single-step SAP Event Stream Processor.

10.8.2.2 Adding Breakpoints

Add breakpoints to Event Stream Processor.

Prerequisites

- Access the Debugger view of the SAP ESP Run-Test perspective
- Enable Trace mode

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 Breakpoints table.
8. To enable, disable, or remove a specific breakpoint, right-click the breakpoint and select an option:
   ○ Enable Breakpoint
   ○ Disable Breakpoint
   ○ Remove Breakpoint
9. To enable or disable all breakpoints, select either Enable All Breakpoints or Disable All Breakpoints.
10. To remove all breakpoints, right-click within the Breakpoints table and select Remove All Breakpoints.
11. Click Trace Off to run Event Stream Processor.
10.8.2.3 Watch Variables

You can 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 Event Stream Processor. When Event Stream Processor runs, the watch table is dynamically updated as run-control events (run, step, pause) are sent through Event Stream Processor.

Related Information

Breakpoints [page 165]
You can insert a breakpoint for any stream in the project.

Adding Breakpoints [page 166]
Add breakpoints to Event Stream Processor.

Adding Watch Variables [page 168]
Add a watch element to a breakpoint.

Pausing SAP Event Stream Processor [page 169]
Pause SAP Event Stream Processor while playing back projects with .rec file types.
10.8.2.4 Adding Watch Variables

Add a watch element to a breakpoint.

Prerequisites

- Access the Debugger view of the SAP ESP Run-Test perspective
- Enable Trace mode

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.

Related Information

Breakpoints [page 165]
You can insert a breakpoint for any stream in the project.

Adding Breakpoints [page 166]
Add breakpoints to Event Stream Processor.

Watch Variables [page 167]
You can insert watch variables into the watch table of the Breakpoints view in the Debugger to inspect data as it flows through the project.

Pausing SAP Event Stream Processor [page 169]
Pause SAP Event Stream Processor while playing back projects with .rec file types.
10.8.2.5 Pausing SAP Event Stream Processor

Pause SAP Event Stream Processor while playing back projects with .rec file types.

Prerequisites

- Access the Debugger view of the SAP ESP Run-Test perspective.
- Enable Trace mode

Procedure

1. In the Debugger, click Pause Project ( ).
2. To resume Event Stream Processor, click Resume Project, or click Trace Off to close the debugger.

Related Information

Breakpoints [page 165]
You can insert a breakpoint for any stream in the project.

Adding Breakpoints [page 166]
Add breakpoints to Event Stream Processor.

Watch Variables [page 167]
You can insert watch variables into the watch table of the Breakpoints view in the Debugger to inspect data as it flows through the project.

Adding Watch Variables [page 168]
Add a watch element to a breakpoint.

Stepping through a Project [page 170]
Single-step SAP Event Stream Processor.
10.8.2.6 Stepping through a Project

Single-step SAP Event Stream Processor.

Prerequisites

- Access the Debugger view of the SAP ESP Run-Test perspective
- Pause the project

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.

Related Information

Breakpoints [page 165]
You can insert a breakpoint for any stream in the project.

Adding Breakpoints [page 166]
Add breakpoints to Event Stream Processor.

Watch Variables [page 167]
You can insert watch variables into the watch table of the Breakpoints view in the Debugger to inspect data as it flows through the project.

Adding Watch Variables [page 168]
Add a watch element to a breakpoint.

Pausing SAP Event Stream Processor [page 169]
Pause SAP Event Stream Processor while playing back projects with .rec file types.
11 Customizing the Studio Work Environment

Customize your Studio interface to work the way you prefer.

Context

Note

As an Eclipse-based application, the SAP Event Stream Processor Studio automatically includes many features not specific to SAP Event Stream Processor. Features documented here have been tested with the SAP Event Stream Processor Studio. Other Eclipse features may not work as expected. For example, the Team Synchronizing perspective is not supported.

11.1 Editing Studio Preferences

Edit preferences to customize the SAP Event Stream Processor environment.

Context

You can also access many of these preferences from the related Studio view.

Procedure

1. Choose Window ➤ Preferences ➤
2. Expand SAP Event Stream Processor, and then expand to the preferences you want to set. All preference settings are optional.

   - **CCL Editor Settings**
     - Set syntax coloring and template options.
   - **Run Test**
     - Set defaults for server connections, add new connections, set limits and filters for the Stream View and Server View, and set other options for running projects in Studio, including the default HANA service entry.
   - **Compiler Options**
     - Change the directory for the CCL compiler output (default is `bin` folder in your `<workspace\project>` folder).
   - **Data Input Settings**
     - Set file upload and SQL Query view options.
Manual Input Settings Choose settings for the publishing data from Manual Input view, including defaults for all datatypes except money types.

Manual Input Settings - Money Types Set defaults for the money(n) datatype.

Network Connections Specify how Studio will connect to other machines on the network.

Shapes General Choose defaults for creating and displaying shapes in diagrams.

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.

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

Related Information

Changing the Display of Diagrams [page 56]
Display diagrams in verbose or iconic mode. Lay out the elements in the diagram left to right or top down.

11.2 Manual Input Settings

Set default values on datatypes for data being published to a stream from the Manual Input view and the format in which the data is published.

Settings for most datatypes are in Manual Input Settings preferences. Settings for the money(n) datatype are in Manual Input Settings - Money Types preferences.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish Multiple Rows</td>
<td>Indicates whether data from an input stream is published in single instances or as multiple rows.</td>
</tr>
<tr>
<td>Use Current Date</td>
<td>Indicates whether data should be published under the current date or maintain its historical date.</td>
</tr>
<tr>
<td>Interpret Date values in Manual Input and Stream Viewer as UTC</td>
<td>Indicates whether Manual Input date values are interpreted as UTC or in the local time zone.</td>
</tr>
</tbody>
</table>

> **Note**
This has no effect on the Playback tool.

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.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>May be set to <em>True</em> or <em>false</em>.</td>
</tr>
<tr>
<td>string</td>
<td>Indicates the default value Studio accepts for string types.</td>
</tr>
<tr>
<td>integer</td>
<td>Indicates the default value Studio accepts for integer types. Does not ac­</td>
</tr>
<tr>
<td></td>
<td>cept values with decimal points.</td>
</tr>
<tr>
<td>float</td>
<td>Indicates the default value Studio accepts for float types.</td>
</tr>
<tr>
<td>long</td>
<td>Indicates the default value Studio accepts for long types.</td>
</tr>
<tr>
<td>interval</td>
<td>Indicates the default value Studio accepts for interval types.</td>
</tr>
<tr>
<td>seconddate</td>
<td>Indicates the default value for seconddate types. Click <em>Select</em> to open a</td>
</tr>
<tr>
<td></td>
<td>calendar dialog and choose a default date with second precision.</td>
</tr>
<tr>
<td>bigdatetime</td>
<td>Indicates the default value for bigdatetime types. Click <em>Select</em> to open a</td>
</tr>
<tr>
<td></td>
<td>calendar dialog and choose a default bigdatetime with microsecond preci­</td>
</tr>
<tr>
<td></td>
<td>sion.</td>
</tr>
<tr>
<td>msdate</td>
<td>Indicates the default value for msdates types. Click <em>Select</em> to open a</td>
</tr>
<tr>
<td></td>
<td>calendar dialog and choose a default timestamp with millisecond precision.</td>
</tr>
<tr>
<td>time</td>
<td>Indicates the default value Studio accepts for time types.</td>
</tr>
</tbody>
</table>

**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 154]

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.
11.3 Rearranging Views in a Perspective

Rearrange the views in a perspective by moving a view to a new docking location in the perspective.

Procedure

1. Click the title bar of the view that you want to move.
2. Hold down the left mouse button and drag the view to the new area.
   As you move the view, the outline is highlighted to help you determine where you can dock the view.
3. When the view is in position, release the left mouse button to drop the view onto the new location.
   When you close the application, the new configuration is saved.
12  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`.

12.1 A Studio Project Does Not Run, Reports Login Failure

Problem: Attempts to run a project in a remote cluster fail with these errors:

```
Failed to connect to server "esp[s]://localhost:19011". Reason: “Failed to login server”
```

Studio reports these errors when it has an SSL mismatch with the ESP server: either SSL is enabled on the server and the Studio connection definition for that server does not include SSL, or the connection definition does include SSL but SSL is not enabled on the server.

Solution: Correct the connection definition in Studio. For details, see Configuring a Remote Cluster Connection in the SAP Event Stream Processor: Studio Users Guide.

12.2 Cannot Publish or Subscribe to a Project that Seems to be Running

Your project appears to be running, but you cannot publish or subscribe to it.

Context

Solution: If there is a single database server running the cluster database, and the server goes offline, you will not be able to connect to the cluster, particularly if authorization is enabled.

Procedure

1. Check the status of the database server, and restart if necessary.
2. If necessary, restart the cluster database using the `dbstart` command.

Note

If authorization is enabled on the cluster database, in the future, try running SQL Anywhere HA option for the cluster database.
3. If you cannot restart the cluster database.
   a. Manually terminate the projects and the cluster manager node using the Linux "kill" command or the Windows Task Manager.
   b. Restart the cluster database.
   c. Restart the cluster manager node.
   d. Restart all projects.

12.3 A Legacy Project Fails to Compile or Run

Problem: A legacy project does not compile or does not run. You might see an error saying there is no such adapter as platform_in or platform_out.

As of Event Stream Processor version 5.1.04, the Platform Input and Platform Output adapters are no longer installed.

Solution: If your project uses platform_in or platform_out to create stream or window bindings in the CCL file, you must remove the bindings from the CCL and recreate them in the CCR file.

1. Remove ATTACH ADAPTER statements for the deprecated adapters from the project:
   ○ ATTACH INPUT ADAPTER platform_in
   ○ ATTACH OUTPUT ADAPTER platform_out
   
   You can do this in Studio using either the visual editor (right-click the adapter element and select Delete) or the CCL text editor (delete the statement). The default location of the CCL file on the Studio machine is <user's-home-dir>/SybaseESP/5.1/workspace/<project-name>/<project-name>.ccl.

2. Recreate each binding in the project configuration (CCR) file. See the SAP Event Stream Processor: Studio Users Guide for information on editing the project configuration. The default location of the CCR file on the Studio machine is the same as the default location for the CCR file.

3. If your old CCL file references adapter parameters that do not appear on the Bindings tab of the Project Configuration File Editor in Studio, call technical support to report the problem.

12.4 Cannot Connect to Server

Problem: When trying to run a project, Studio returns a connection error.

Context

When running a project, the following error message appears:

<Cannot connect to server>

The error message and the Studio console do not explain the source of the error; it may be caused by an invalid license key.
Solution: View the Studio log file to find out the source of the error.

Procedure

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 installation documentation or the SAP Event Stream Processor: Configuration and Administration Guide.

12.5 Cannot Connect to the Cluster

When running a project, you cannot connect to the SAP Event Stream Processor cluster.

Context

Solution: If there is a single database server running the cluster database, and the server goes offline, you will not be able to connect to the cluster, particularly if authorization is enabled.

Procedure

1. Check the status of the SAP Event Stream Processor server, and restart if necessary.
2. If necessary, restart the cluster database using the dbstart command.

   i Note
   If authorization is enabled on the cluster database, in the future, try running SQL Anywhere HA option for the cluster database.

3. If you cannot restart the cluster database,
   a. Manually terminate the projects and the cluster manager node using the Linux “kill” command or the Windows Task Manager.
   b. Restart the cluster database.
   c. Restart the cluster manager node.
   d. Restart all projects.
12.6 Cannot Open a Project in Studio

Problem: Attempts to open a project fail and cause errors.

Context

You encounter the following error:

```
Could not open the editor: Invalid thread access
```

This problem 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.

Procedure

1. From the command line, navigate to the ESP install directory. The default name of this directory is /Streaming.
2. Set the environment variables:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Run STREAMING.bat</td>
</tr>
<tr>
<td>UNIX</td>
<td>Source STREAMING.csh, STREAMING.sh, or STREAMING.env</td>
</tr>
</tbody>
</table>

3. Navigate to <install-directory>/ESP-5_1/studio.
4. Run esp_studio.exe to start Studio.

12.7 Changing Date Format for Playback

Problem: Unable to load or playback files containing dates in formats other than UTC.

Context

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.
Procedure

1. Open the SAP ESP 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. Refer to Running and Testing a Project > Playback View for more information.

Next Steps

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 Event Stream Processor: Utilities Guide.

12.8 An External Adapter Fails to Start

Problem: Attempts to start an external adapter fail.

Context

You encounter an error message like this when attempting to run an external adapter:

```
Failed call to:https://<ESP hostname>:61308/RPC2 (Failed to read server's response: <ESP hostname>) java.io.IOException: Failed call to:https://<ESP hostname>:61308/RPC2 (Failed to read server's response: <ESP hostname>)
```

This error is an example of the Event Stream Processor 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:

Procedure

1. Determine the IP address of the host on which the server is running. For example, if you want to determine the IP address of the Event Stream Processor server host, run this command from that machine:

   `nslookup <ESP hostname>`

2. Add the following line to `C:\Windows\System32\drivers\etc\hosts` (Windows) or `/etc/hosts` (UNIX):

   `<ipaddress of server hostname> <Server hostname>`
12.9 Prompted for Local Cluster Password

Problem: While connecting to the Studio local cluster from a component outside of Studio (such as command line tools, external adapters, or custom applications), you are prompted for a password.

Context

Solution: The Studio local cluster password is system-generated by default. In order to enter this password, you will first have to change it.

Procedure

1. In Studio, open the SAP ESP Run-Test perspective.
2. In the Server View, stop any projects running in the local cluster.
3. Right-click the local cluster, and select Disconnect Server.
4. Right-click the local cluster, and select Change Username and Password.
5. Enter the new values for the local cluster user name and password.
   Ensure that you make note of the new values as you will be prompted to enter them every time you connect to the local cluster with a component from outside of SAP ESP Run-Test perspective.
6. Right-click the local cluster, and select Connect Server.

12.10 Playback is Too Fast or Slow

Problem: While using the playback tool in the SAP ESP Run-Test perspective, data plays back too fast or too slow.

Context

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.
Procedure

1. Open the 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.

12.11 Studio Crashes and You Are Unable to Restart It

Problem: The Studio workspace may have become corrupt.

Context

Solution: Migrate your projects to a new workspace.

Procedure

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 ESP Authoring perspective, import your projects into the new workspace. See Importing Multiple
   Projects [page 33] or Importing a Project into SAP HANA Studio [page 36] for detailed steps.

12.12 Retention Policy Errors

Problem: In a project, windows without retention policies cause errors, and the project cannot compile or run.

Context

When compiling or running a project, any window without a retention policy causes one or both of the following
error messages:
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. To fix this issue:

**Procedure**

1. Ensure that STREAMING_HOME is set as a system variable.
   a. Go to Control Panel, then select \Control Panel\ System \ Advanced System Settings \ Environment Variables. 
   b. Set the STREAMING_HOME variable to `<install dir>`\ESP-5_1.
2. Restart Studio.

### 12.13 Stream View Causes Project to Run Slower

**Problem:** Project runs slower when streams or windows are open in the Stream View.

**Context**

As it tries to keep up with the data flowing through the server, Studio blocks incoming data and slows down the project.

**Solution:** Enable lossy subscription. This option allows Studio’s subscription to lose data if it is unable to keep up with the data flowing through the server.

**Procedure**

1. Go to Window \ Preferences \ SAP Event Stream Processor \ Run Test.
2. Select Streamviewer lossy subscription. Click OK to save.
12.14 Stream View Displays Partial Data

Problem: In the SAP ESP Run-Test perspective, the Stream View does not show all rows of a running project.

Context

Solution: Increase the number of visible rows in the Stream View preferences.

Note

Increasing the number of rows will also increase Studio’s memory usage.

Procedure

1. Open the SAP ESP 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.

Next Steps

See Viewing a Stream for more information on Stream View preferences.

If Stream View does not show any data, the data may have already been processed. See Stream View Does Not Show Any Data for more information.

12.15 Stream View Does Not Display Any Data

Problem: When running a project that uses one or more file input adapters, Stream View does not show any data.

Context

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.

**Procedure**

1. In the SAP ESP Authoring 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 ESP Run-Test perspective, open the necessary streams in Stream View.
4. Open the File Upload tool to manually load the data.
   
   See *Uploading Data to the Server* to learn how to load data into a project.

**Next Steps**

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* for more information.

### 12.16 An Adapter Fails to Connect to a Project

**Problem:** When attempting to start an adapter without editing its sample XML configuration file, the adapter fails to start.

**Context**

The adapter may be unable to connect to the example workspace specified in the sample XML adapter configuration file if you have SSL enabled on the cluster and the URI specified in the file uses esp instead of esps. The mismatch between the cluster and adapter configuration would cause the adapter to fail to connect.

**Solution:** If SSL is enabled on the cluster on which you are attempting to run this adapter, follow the steps below.

**Procedure**

1. Ensure the URI in the adapter XML configuration file uses esps instead of esp.
2. If attempting to run 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
   ```
12.17 An External Adapter Fails to Start

Problem: Attempts to start an external adapter fail.

Context

You encounter an error message like this when attempting to run an external adapter:

```
Failed call to:https://<ESP hostname>:61308/RPC2 (Failed to read server's response: <ESP hostname>)
java.io.IOException: Failed call to:https://<ESP hostname>:61308/RPC2 (Failed to read server's response: <ESP hostname>)
```

This error is an example of the Event Stream Processor 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:

Procedure

1. Determine the IP address of the host on which the server is running. For example, if you want to determine the IP address of the Event Stream Processor server host, run this command from that machine:

   `nslookup <ESP hostname>`

2. Add the following line to `C:\Windows\System32\drivers\etc\hosts` (Windows) or `/etc/hosts` (UNIX):

   `<ipaddress of server hostname>        <Server hostname>`
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 willful 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).