Studio Users Guide

Sybase Event Stream Processor

5.1

DOCUMENT ID: DC01613-01-0510-01
LAST REVISED: July 2012

Contents

CHAPTER 1 Introduction to Sybase Event Stream Processor

Sybase® Event Stream Processor enables you to create and run your own complex event processing (CEP) applications to derive continuous intelligence from streaming event data in real time.

Event Stream Processing and CEP

Event stream processing is a form of 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 processing, you can analyze events as they stream in and identify emerging threats and opportunities as they happen. Event Stream Processor Server filters, aggregates, and summarizes data to enable better decision making based on more complete and timely information.

Event Stream Processor 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.

Event Stream Processor as a Development Platform

As a platform for developing CEP projects, 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.

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. Event Stream Processor:

- Processes data continuously as it arrives
- Processes data before it is stored on disk, 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

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.

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
- 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

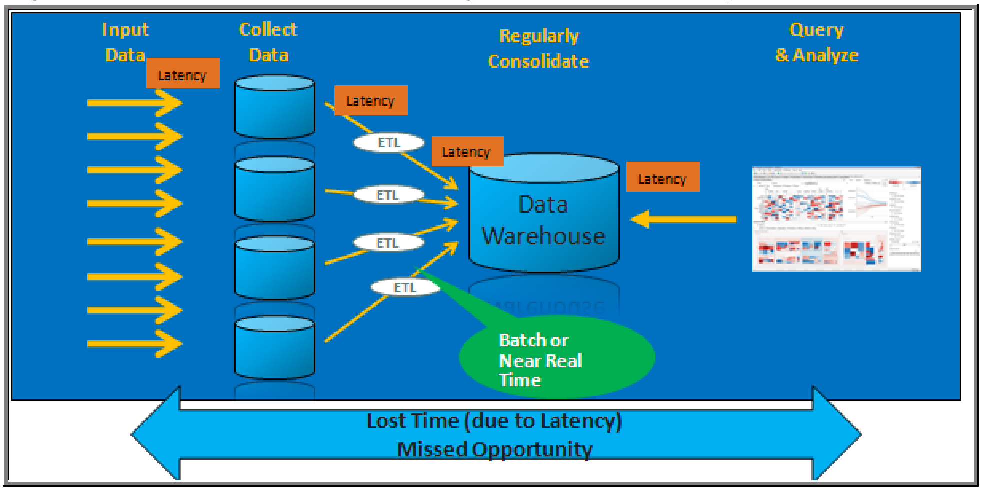
Event Stream Processor Compared to Databases

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

Event Stream Processor executes queries continuously on fast moving data streams.

Event Stream Processor is not a replacement for databases. While databases excel at storing and querying static data, and reliably processing transactions, they are not effective at continuously analyzing fast moving streams of data.

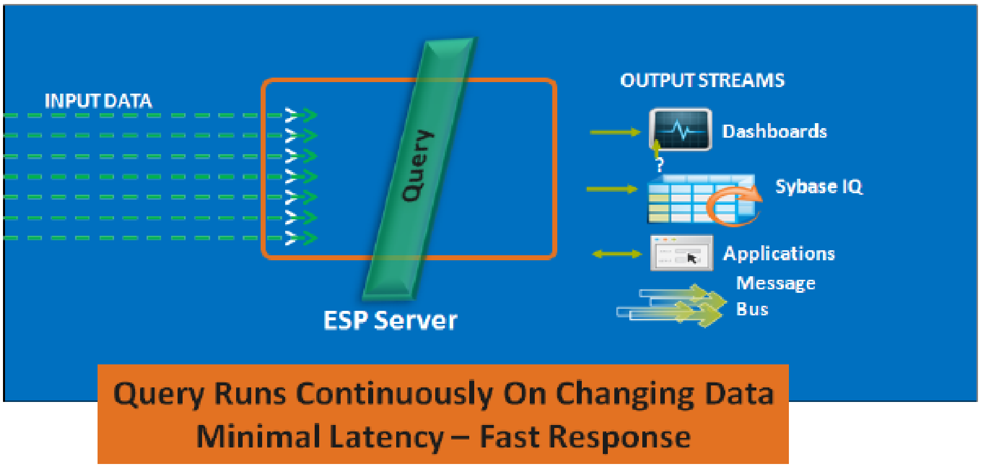
Figure 1: Traditional Business Intelligence: On-Demand Queries



- Traditional databases must store all data on disk before beginning to process it.
- Databases do not use preregistered continuous queries. Database queries are "one-time-only" queries. To ask a question ten times a second, you must issue the query ten times a second. This model breaks down when one or more such queries need to be executed continuously, as polling the database faster results in a performance impact to the source systems, and adds latency.
- Databases do not use incremental processing. Event Stream Processor can evaluate queries incrementally as data arrives.

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

Figure 2: Continuous Queries in Event Stream Processor



Data-Flow Programming

Sybase® 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.

Figure 3: Data-Flow Programming - Simple Example

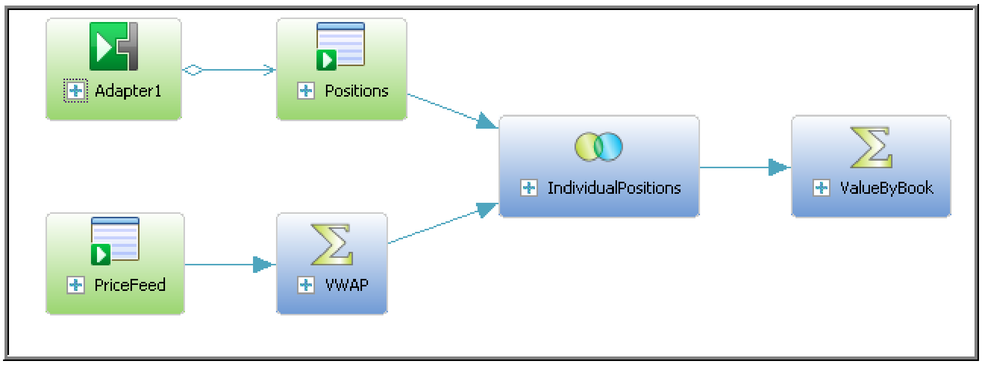
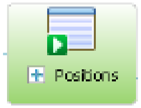
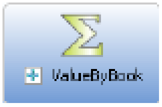


Table 1. Data-Flow Diagram Contents

Element	Description
PriceFeed	Represents an input window, where incoming data from an external source complies with a schema consisting of five columns, similar to a database table with columns. The difference is that in ESP, the streaming data is not stored in a database.

Element	Description
Positions 	Another input window, with data from a different external source. Both Positions and PriceFeed are included as windows, rather than streams, so that the data can be aggregated.
VWAP	Represents a simple continuous query that performs an aggregation, similar to a SQL Select statement with a Group By clause.
IndividualPositions	Represents a simple continuous query that performs a join of Positions and VWAP, similar to a SQL FROM clause that produces a join.
ValueByBook 	Another simple query that aggregates data from the stream Individual Positions.

ESP Projects: Streams, Windows, Adapters, and Continuous Queries

An ESP project is like an application, consisting 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:

- **Input streams and windows –** where the input data flows 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.
- **Adapters –** connect an input stream or window to a datasource. Sybase Event Stream Processor 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.
- **Derived streams and windows –** take data from one or more streams or windows and apply a continuous query to produce a new stream or window. Derived streams that have state are windows.

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 visual authoring palette lists local and output streams as derived streams, and lists local and output windows as derived windows.

Getting Results from an ESP Project

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

- Applications receive information automatically from internal output adapters attached to a stream when you build the project.
- Applications can subscribe to data streams by means of an external subscriber, which users can create using subscription APIs provided with the product.
- Users can start a new project that binds (connects) to a stream in a running project, without reconfiguring the project.
- Users 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 esp_query tool. For more information see the *Utilities Guide*.
- In ESP Studio, using the SQL Query view tools.

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).

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. The ESP Server 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, or 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 cancelled 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 (stream, window, or delta stream) and the processing specified for it.

Refer to the *Streams*, *Windows*, and *Delta Streams* topics in the *Programmers Guide* for details on how each interprets the opcodes on incoming event records and generates opcodes for output records.

Product Components

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:

- **ESP Server –** the software that processes and correlates data streams at runtime. Event Stream Processor can process and analyze hundreds of thousands of messages per second. Clustering provides scale-out support to ESP 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.
- **ESP Studio –** an integrated development environment for creating, modifying, and testing ESP projects.
- **CCL compiler –** the compiler that translates and optimizes projects for processing by ESP Server. It is invoked by ESP Studio or from the command line.
- **Input and output adapters –** the components that establish connections between Event Stream Processor and datasources, as well as the connections between the ESP Server and the consumers that will receive output from Event Stream Processor.
- **Integration SDK –** 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.
- **Utilities –** a set of executables that offer command line access to many administrative, project development, publishing and subscription, and other features.

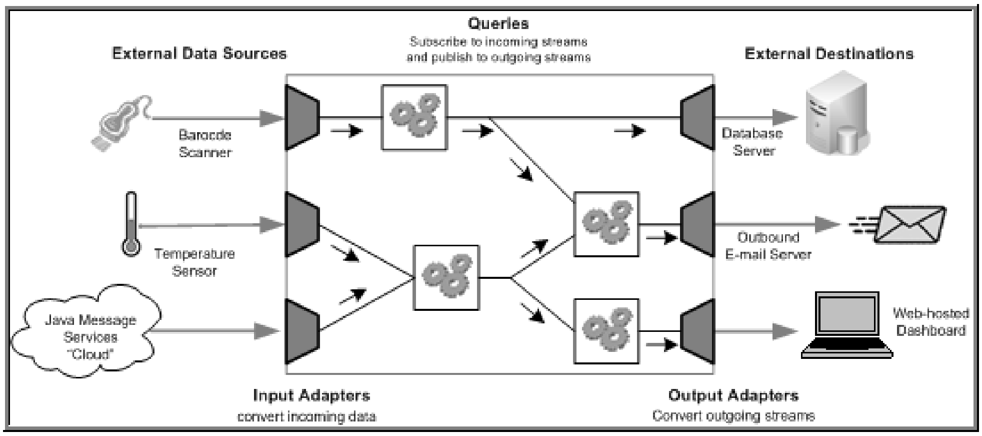
Do not mix components from different versions of Event Stream Processor. For example, do not run ESP Server from the current version along with ESP Studio from the previous version.

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.

Figure 4: Adapters in Event Stream Processor



For a complete list of adapters supplied by Event Stream Processor, see the *Adapters Guide*.

Custom Adapters

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

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#, Visual Basic, and so on)

For detailed information about how to create custom adapters, see the *Adapters Guide*. For versions supported by these SDKs, see the *Installation Guide*.

Authoring Methods

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. You can also compile projects within Studio.

In addition to its visual and text authoring components, Studio includes environments for working with sample projects, and for running and testing applications with a variety of debugging tools. Studio also lets you record and playback project activity, upload data from files, manually create input records, and run ad hoc queries against the server.

If you prefer to work from the command line, you can develop and run projects using the **esp_server**, **esp_client**, and **esp_compiler** commands. For a full list of Event Stream Processor utilities, see the *Utilities Guide*.

Continuous Computation Language

CCL is the primary event processing language of the Event Stream Processor. ESP projects are defined in CCL.

CCL is based on Structured Query Language (SQL), adapted for event 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 on 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 an 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 Event Stream Processor Studio, but it can also be performed by invoking the CCL compiler from the command line.

SPLASH

Stream Processing LAnguage SHell (SPLASH) 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 SPLASH scripts in CCL provides tremendous flexibility, and the ability to do it within the CCL editor maximizes user productivity. SPLASH also allows you to define any complex computations that are easier to define using procedural logic rather than a relational paradigm.

SPLASH 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. SPLASH 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.

See also

- *Flex Operators* on page 62

CHAPTER 2 Getting Started in ESP Studio

To begin developing a project, start ESP Studio, review workspace basics, and optionally step through an example before creating your own project.

Starting ESP Studio

Start ESP Studio from the desktop shortcut, Windows Start menu, or the command line. From your desktop or workstation:

Platform	Method
Windows	• Double-click the **Sybase ESP Studio** shortcut on your computer desktop, or, • Select **Start** > **Programs** > **Sybase** > **Event Stream Processor 5.1** > **Studio** > **Studio**.
Linux or UNIX	• Double-click the **Sybase ESP Studio** shortcut on your computer desktop, or, • At the command line, enter `$ESP_HOME/studio/esp-studio`.

Studio Workspace Basics

In the Studio workspace, 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.

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

Perspective	Activities
Authoring	• Create and edit projects • Develop projects and diagrams in the Visual editor, a graphical editing environment • Develop projects in the CCL editor, a text-oriented editing environment where you edit CCL code • Compile projects • Import Aleri models
Learning	• Load example projects • Step through example projects so that you can follow what happens when you subscribe to streams, publish demonstration data, and view results **Note:** Activities you initiate in Learning perspective open in Authoring and Run-Test perspectives so that you can take advantage of facilities there to learn more about the example project.
Run-Test	• Start and connect to servers • Run projects • Enter test data by uploading data files to a server, or entering data manually to a stream • Publish data • Execute a query against a running project • Use the Event Tracer and Debugger to set breakpoints and watchpoints, and trace the flow of data through a project • Record incoming event data to a playback file, and play back captured data into a running project • Monitor performance

See also

- *Chapter 3, Visual Editor Authoring* on page 21
- *Chapter 7, Running and Testing a Project* on page 93

File Explorer

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

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

- Creating new CCL files

- Creating new projects
- Editing existing files
- Deleting files
- Creating new folders

See also

- *Editing a Project in the Visual Editor* on page 27
- *Editing in the CCL Editor* on page 67

The Studio Log File

The 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`, but you can view the log within Studio. To view the log:

1. Select **Help** > **About Studio**.
2. Click **Configuration Details**.
3. Click **View Error Log**.
4. 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.

Learning Perspective

The Learning perspective helps you get started with Studio by performing common tasks with example projects.

You can open the Learning perspective in three different ways:

- The **Open Example** shortcut on the Welcome Screen
- The **Learning** button on the perspective shortcuts bar
- Select **Window** > **Open Perspective** > **Learning**

The Examples view, which appears on the left in the Learning perspective, lists all currently available examples. Each example item has a **LOAD** button, which you can click to run the example.

The Description view, to the right of the Learning perspective, shows a detailed description of the selected example.

Running Examples in the Learning Perspective

Load and run provided examples to demonstrate Server View, Stream View, Visual editor and other important Studio functions.

1. Select an example project from the Examples view and click **LOAD**.
2. Either:
    - Click **Proceed** to start the example project and follow prompts for the remaining steps; or,
    - Check **Run in silent mode** and click **Proceed** to run the process in the background.

    Once this step is complete, the example project appears in the Visual editor.
3. Click **Proceed** to subscribe to the example output and view it.
   The stream appears in Stream View.
4. Click **Proceed** to publish the example data and upload it to a server.
5. (Optional) In the Step by Step Example view, select a project from the **Example project** menu.

    If the project has the same name as an existing project in your workspace, Studio determines whether or not the existing project is also an example. If it is, studio loads the project. If the existing project is not an example, an error results and you must either rename the example project you initially selected, or remove the existing project that caused the error.
6. (Optional) Click a step to review any of the actions previously described.

    The Step by Step Example view lists the actions that the example performs and provides a quick link for launching the actions. The project must be running to launch an action.

Creating a Project

Use the Studio to create new projects that can run on the ESP Server.

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

1. Select **File > New > 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.

Studio 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," Studio 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.

See also

- *Opening a Project* on page 18
- *Importing an Existing Project* on page 19
- *Editing a Project in the Visual Editor* on page 27
- *Switching Between the CCL and Visual Editors* on page 65

Converting AleriML Models into CCL Projects

Studio allows you to convert AleriML data models into new CCL projects, or add the data to an existing project.

Any conversion errors appear in a dialog box, wherein each error appears as a separate row, along with line and column information.

See the *Migration Guide* for differences between Aleri models and ESP projects.

Converting AleriML Models into New Projects

Access AleriML conversion functionality for new projects in the File menu.

1. From any view in Studio, open the **File** menu.
2. Select **Convert Aleri Data Model**. Select whether to convert the data model into a new CCL project or add the data file to an existing project.
3. Select **Convert to new project** and click **Next**.
4. Browse to or enter the name of the **Aleri data model** you wish to convert. The **CCL file name** and **Project name** fields are populated based on the model you select.

   You can overwrite the CCL file and project names after the fields are populated.
5. Accept the default **Location** or browse to a directory in which to store the new project folder.
6. Click **Finish** to complete the conversion.

See also

- *Converting AleriML Models to Add to Existing Projects* on page 18

Converting AleriML Models to Add to Existing Projects

Access AleriML conversion functionality for existing projects from the File menu or File Explorer.

1. Either:
    - Open the **File** menu from any perspective in the Studio, or,
    - Right-click a project in the **File Explorer** view in the Authoring perspective.
2. Select **Convert Aleri Data Model**. If you accessed the conversion option from the **File** menu, select **Convert to existing project** and click **Next**.

    Note: Your project must be located in the current workspace.
3. Browse to or enter the name of the **Aleri data model**.

    The **CCL file name** and **Project name** fields are populated based on the name of the model.
4. Click **Finish** to complete the conversion.

See also

- *Converting AleriML Models into New Projects* on page 17

Opening a Project

Open an Event Stream Processor project from File Explorer when it already exists in your workspace.

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

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

See also

- *Creating a Project* on page 16
- *Importing an Existing Project* on page 19
- *Editing a Project in the Visual Editor* on page 27
- *Switching Between the CCL and Visual Editors* on page 65

Importing an Existing Project

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

1. Choose **File > Open > 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**.

See also

- *Creating a Project* on page 16
- *Opening a Project* on page 18
- *Editing a Project in the Visual Editor* on page 27
- *Switching Between the CCL and Visual Editors* on page 65

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.

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.

1. In the **Authoring** perspective, right-click the **File 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**.

CHAPTER 3 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.

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.

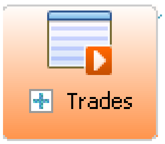
When you open a project in the Visual editor, the project shows a collection of stream and window shapes that are connected with arrows showing the flow of data. You develop the project by selecting new input and output streams, windows, and other elements from the Palette, dropping them onto the diagram,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

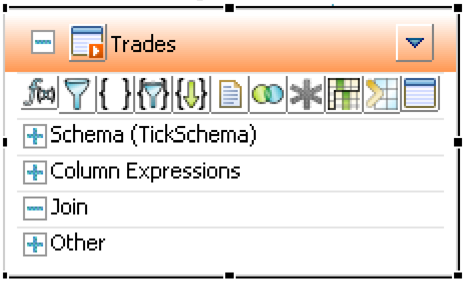
When you add a shape or other element to a diagram, it is automatically added to the project when you save. You can delete an element from a diagram only, or from the project.

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.
- To collapse an individual shape, select it and click the "-" box in the shape header.

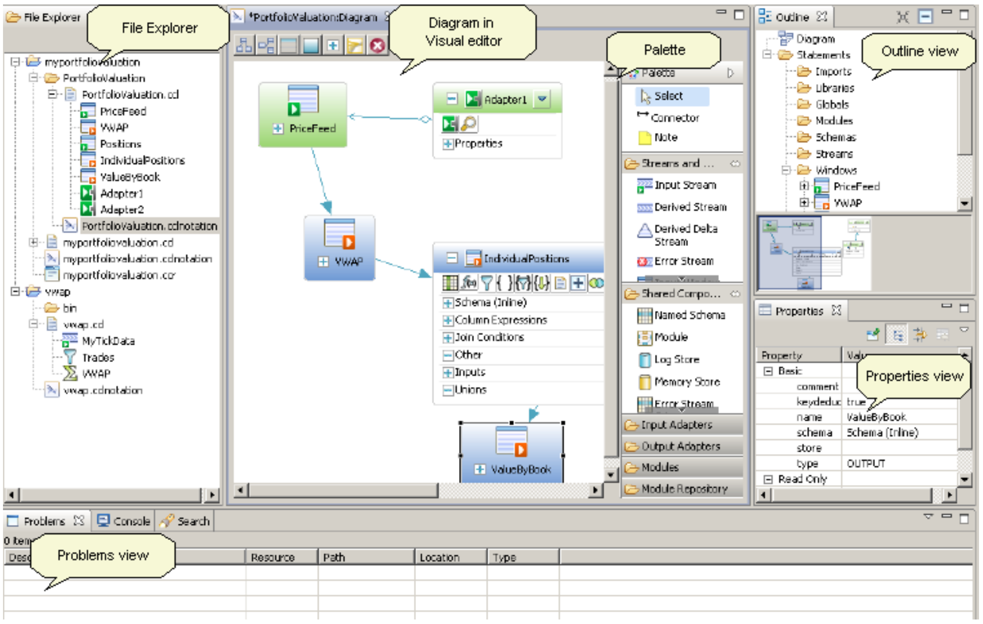
See also

- *Shape Reference* on page 24
- *Changing the Display of Diagrams* on page 30

Studio Authoring Views and Editors

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

Figure 5: Authoring Perspective Views



- **Editor –** canvas at the center of the 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 –** includes groups of tools used to create new CCL elements on the diagram. Most shapes on the Palette correspond to a CCL statement.
- **File Explorer –** provides a hierarchical tree structure of folders and files.
- **Properties view –** displays the properties of the object selected in the diagram. You can also set properties in this view, and edit expressions.
- **Outline view –** provides an index to all elements in the diagram as a hierarchical tree structure. Also shows the order in which adapters are started. Right-click an element in this view to show it in the diagram, delete it, modify it, or add a child element.
- **Overview –** helps you understand the big picture, and navigate easily to different areas of a large, complex diagram. For large diagrams you can scroll the editor by dragging the gray box in the overview.
- **Search –** provides full-text search capability for finding text strings in the workspace. Useful in navigating File Explorer, and project contents in the CCL editor. You can filter search results, and copy, remove, or replace results found.
- **Problems –** displays errors found when you compile a project or convert an Aleri model to CCL.
- **Console –** displays messages generated when interacting with ESP components.

Note: ESP Studio lets you customize the arrangement of views in your perspectives. See *Customizing the Studio Work Environment* in the *Studio Users Guide*.

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

Shape	Purpose	Usage
Connector	Creates flows between streams and windows, establishes references between streams and shared components, or attaches notes to shapes.	Click to select the connector tool, then click each of the shapes in the diagram to be connected..
Note	Creates a comment on the diagram only. This comment does not appear in the CCL file.	
Input Stream	The entry point for unkeyed event streams into a project. Receives data from either an input adapter or an external publisher.	A stream does not retain any data and does not have a state. Data in an input stream is not keyed.
Derived Stream (Local) Derived Stream (Output)	Applies a continuous query to data arriving from another stream or window to produce a new stream.	Streams do not retain data and do not have keys. They are "insert only," meaning that their output consists only of inserts. Input must be a stream or a stream-window join. By default, new streams (including derived streams) are local, but you can change that property to output, to make them visible to external subscribers.

Shape	Purpose	Usage
Derived Window (Local) Derived Window (Output)	Applies a continuous query to data arriving from another stream or window. Retains data, and retention rules can be set.	Data must be keyed so that every row has a unique key. Processes inserts, updates, and deletes both as local and output. You can use the toolbar to change the window to ouput, which makes it visible to external clients.
Derived Delta Stream (Local) Derived Delta Stream (Output)	Applies a continuous query downstream from a window where there is no need to retain state but there is a need to preserve insert, update, and delete operations.	Can be used where a computation, filter, or union must be performed, but where a state does not need be maintained. Use the toolbar to change the derived delta stream to output if needed.
Input Window	The entry point for event streams into a project where incoming events have primary keys and there is a desire to maintain a window of event data. Supports opcodes (insert, update, delete, upsert). Use this as an entry point for event streams if: • The stream contains insert, update and delete events, or, • You need to retain a set of incoming events.	Window size can be set by row count with a fixed number of input records, or by time with records being kept for a specified period. The window must be keyed, that is, every row must have a unique key value.
Flex	A programmable operator that uses custom SPLASH scripts to process incoming events.	A Flex operator can take input from any number of streams and/or windows and will produce a new derived stream or window (either local or output).
Aggregate	Takes input from a single stream or window and groups records using a common attribute. Produces a single output record for each group. Uses aggregate functions like sum(), count(), and so on.	Always creates a new window. Requires a GROUP BY element. You can optionally set window size using retention rules.

Shape	Purpose	Usage
Compute	Takes input from a single source and computes a new record for every record received. Allows you to change the schema on events, computing new fields and changing existing fields.	Produces a derived stream when the input is a stream. Produces a derived delta stream when the input is a window.
Filter	Takes input from a single source and applies a filter. Creates a stream of records that match the filter criteria.	Produces a derived stream when the input is a stream. Produces a derived delta stream when the input is a window.
Join	Takes input from two or more sources and joins them based on common data elements.	See related information in this guide and the *CCL Programmers Guide* for join support details.
Pattern	Takes input from two or more sources and detects patterns of events. One output record is produced every time a pattern is detected.	
Union	Merges input from two or more sources. One ouput record is produced for every input record.	All inputs must have a common schema.
Named Schema	Reusable definition of column structure that can be referenced by streams and windows.	
Module	Represents a CCL **CREATE MODULE** statement. Creates a new module that can be used in one or more places in the project.	A module can contain all the same elements as a project and provides for reuse.
Log Store	Stores data held in windows. Provides disk-based recovery but is slower than a memory store	By default, new windows are assigned to a memory store. Where recoverability of data in a window is required, create a log store and assign the window to it.

Shape	Purpose	Usage
Memory Store	Stores data held in windows.	Faster than a log store but does not recover data after shutdown. • (Default) Created implicitly by the CCL compiler, if no other store is specified. • (Optional) Created explicitly, with windows assigned to specific stores, to optimize performance.
Input Adapters	Connects an input stream or input window to an external data source.	Must be connected to either an input stream or input window. To use schema discovery—that is, to import the schema from the source—add the input adapter first, and then use schema discovery to create a connected input stream or window with the imported schema.
Output Adapters	Connects an output stream or window to a destination.	Must be connected to either an output stream or an output window.

See also

- *Simple Queries* on page 38
- *Adding Shapes to a Diagram* on page 28
- *Connecting Elements* on page 48
- *Join Types and Restrictions* on page 44

Editing a Project in the Visual Editor

Edit diagrams in a graphical user interface.

1. In the Authoring perspective, navigate to **File 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 .

Note: The Visual editor, like other graphical user interfaces, offers several ways to accomplish most tasks, although this guide may not list all of them. For example, in many contexts you can carry out an action by:

- Clicking a button or other icon in a shape, or on the main toolbar
- Using a shortcut key
- Double-clicking an element to open it
- Right-clicking to select from the context menu
- Selecting from the main menu bar
- Editing element values in the Properties view

ESP Studio also includes features common to Eclipse-based applications.

See also

- *Creating a Project* on page 16
- *Opening a Project* on page 18
- *Importing an Existing Project* on page 19
- *Switching Between the CCL and Visual Editors* on page 65
- *File Explorer* on page 14

Adding Shapes to a Diagram

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

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.

   Note: Do not try to drag-and-drop from the Palette into the diagram.
3. To view actions needed to complete a shape definition, hover the mouse over the shape in the diagram.

Next

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

See also

- *Simple Queries* on page 38
- *Shape Reference* on page 24
- *Deleting an Element* on page 51
- *Keyboard Shortcuts in the Visual Editor* on page 29

Adding Comments to Shapes

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

Prerequisites

'Show comments in tooltip' must be enabled in Preferences.

Task

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.

Keyboard Shortcuts in the Visual Editor

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

This table lists commonly used keyboard shortcuts. For a complete list, choose **Help > Key Assist** (**Ctrl+Shift+L**).

Key	Action
F2	Edit the selected shape name or element within a shape (context dependent)
F4	Toggle between CCL editor and Visual editor
F7	Compile
F11	Toggle between Authoring and Run-Test perspectives
Insert	Insert new item to a compartment
Delete	Delete selected elements from project

Key	Action
Ctrl +Delete	Delete selected elements from diagram
Ctrl + A	Select all
Ctrl + N	Open a new project
Ctrl + Y	Redo
Ctrl + Z	Undo
Ctrl + F2	Open column expression editor
Ctrl + Space	Show available columns and built-in functions for column expression
Ctrl + Mouse wheel	Zoom in or zoom out
Ctrl + Shift + L	List all keyboard shortcut assignments
Alt + U	Move compartment item up in the Outline
Alt + D	Move compartment item down in the Outline
Alt + T	Toggle shape between iconic and verbose mode

See also

- *Adding Shapes to a Diagram* on page 28
- *Deleting an Element* on page 51

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.

- 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.

See also

- *Editing Studio Preferences* on page 111

Building a Simple Project

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

Prerequisites

Create the project.

Task

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*

   Attach an adapter by inserting it in the diagram, connecting it to a stream or window, and setting properties.
2. *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.
3. *Adding an Input Stream or Window to a Project*

   Input streams and windows accept data from a source external to the project.
4. *Adding a Simple Query*

   Choose the type of simple query you want and use the tools in the Visual editor to create it.
5. *Connecting Elements*

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

   Set primary keys in the Visual editor within the Column compartment of the delta stream, window, and Flex operator shapes.
7. *Editing Column Expressions for Windows, Streams, and Delta Streams*

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

See also

- *Deleting an Element* on page 51
- *Creating a Project* on page 16

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.

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 **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,
   - Select **Set properties locally** to manually configure the adapter properties.

See also

- *Discovering a Schema* on page 33

Schema Discovery

You can use the schema discovery feature to discover external schemas and create CCL schemas based on the format of the data from the datasource 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 may use the same schema, but a stream or window can only have one schema.

Rather than manually creating a new schema, you can use schema discovery to discover and automatically create a schema based on the format of the data from the datasource connected to your adapter. For example, for the Database Input adapter, you can discover a schema that corresponds to a specific table from a database the adapter is connected to.

While using discovery is a convenient way to create your CCL schema, pay particular attention to the datatypes your CCL columns inherit from the external data source. 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 Sybase IQ, support microsecond precision for the `SQL_TIMESTAMP` and `SQL_TYPE_TIMESTAMP` data types. As such, 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 `timestamp` (millisecond precision).

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

See also

- *Appendix A, Adapter Support for Schema Discovery* on page 115
- *Discovering a Schema* on page 33

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

Add the adapter to the diagram.

Task

In the Authoring perspective:

1. Configure the adapter for schema discovery. In the adapter shape, click **Edit Properties** and complete the dialog:

   - Select a named property set, or,
   - Choose **Set properties locally** 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.

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

5. Click **Finish**.

See also

- *Schema Discovery* on page 32
- *Appendix A, Adapter Support for Schema Discovery* on page 115
- *Adding an Adapter to a Project* on page 32

Adding an Input Stream or Window to a Project

Input streams and windows accept data from a source external to the project.

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.

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.
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**.

   Note: 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.

See also

- *Adding a Simple Query* on page 38
- *Specifying a Retention Policy* on page 35
- *Editing Column Expressions for Windows, Streams, and Delta Streams* on page 49
- *Setting an Aging Policy* on page 63

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

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

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

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	Description
PER clause	(Optional) Works with the Time and Count options. When used with the Count option, PER works in conjunction with the specified Count number to retain the Count number of rows across each column specified under the PER clause. When used with the Time option, PER works in conjunction with the specified Time interval to retain the rows within that Time interval across each column specified under the PER clause. List the names of the columns that need to be retained in the PER clause box, with a comma separating each column name entered.

Count

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

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

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

Slack

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

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

Note: 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 becomes 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.

See also

- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Join* on page 43

Adding a Simple Query

Choose the type of simple query you want and use the tools in the Visual editor to create it.

See also

- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Compute* on page 42
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Union* on page 46
- *Creating and Modifying Simple Queries: Pattern* on page 47
- *Simple Queries* on page 38
- *Adding an Input Stream or Window to a Project* on page 34
- *Connecting Elements* on page 48

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 Server 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)

Simple Query	CCL
Filter	WHERE clause
Aggregate	GROUP BY clause
Join	FROM clause, WHERE clause, ON clause
Compute	Simple SELECT statement, with column expressions
Union	UNION clause
Pattern	MATCHING clause

Simple Queries from CCL Statements

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

```
/**@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 delta stream.
- 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 delta stream.
- 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 delta stream.
- All pattern objects create a stream.

See also

- *Shape Reference* on page 24
- *Adding Shapes to a Diagram* on page 28
- *Connecting Elements* on page 48
- *Queries in CCL* on page 69
- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Compute* on page 42
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Union* on page 46
- *Creating and Modifying Simple Queries: Pattern* on page 47

Creating and Modifying Simple Queries: Filter

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

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.

See also

- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Compute* on page 42
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Union* on page 46
- *Creating and Modifying Simple Queries: Pattern* on page 47
- *Simple Queries* on page 38

Creating and Modifying Simple Queries: Aggregate

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

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:
   a) Click **Copy Columns from Input** () in the shape toolbar to select the columns to copy into the schema for the Aggregate window.
   b) Add additional columns by clicking **Add Column Expression** in the shape toolbar.
   c) Edit a column expression by double-clicking to open the inline editor, or by selecting the expression and pressing **Ctrl+F2** to open it in the pop-up editor.
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 change the aggregate object from LOCAL to OUTPUT.

See also

- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Compute* on page 42
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Union* on page 46
- *Creating and Modifying Simple Queries: Pattern* on page 47
- *Simple Queries* on page 38
- *Specifying a Retention Policy* on page 35

Creating and Modifying Simple Queries: Compute

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.

1. In the Visual editor Palette, in **Streams and Windows**, select **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 **Copy Columns from Input** () in the shape toolbar to copy input fields into the schema for this query.
   b) Add additional columns by clicking **Add Column Expression** in the shape toolbar.
   c) Edit a column expression by double-clicking to open the inline editor, or by selecting the expression and pressing **Ctrl+F2** to open it in the pop-up editor.
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 LOCAL or INPUT.

See also

- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Union* on page 46
- *Creating and Modifying Simple Queries: Pattern* on page 47
- *Simple Queries* on page 38

Creating and Modifying Simple Queries: Join

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

1. In the Visual editor Palette, in **Streams and Windows**, select **Join** ( ).
2. Select a location in the diagram and click to add the shape.
3. Connect the join object to the streams or windows that provide the inputs to the join.

   Connect join objects to two or more streams, windows, or Flex operators. Join objects can take input from two or more objects, but can produce only one output.

   Note: Streams, windows and 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 *CCL Programmers 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, or by selecting the expression and pressing **Ctrl+F2** to open it in the pop-up editor.

   Or, edit the corresponding fields in the Properties view.
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.
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.

See also

- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Compute* on page 42

- *Creating and Modifying Simple Queries: Union* on page 46
- *Creating and Modifying Simple Queries: Pattern* on page 47
- *Simple Queries* on page 38
- *Specifying a Retention Policy* on page 35
- *Join Types and Restrictions* on page 44

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 ESP Studio, you must 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.

Event Stream Processor supports all join types:

Join Type	Description
Inner Join	One record from each side of the join is required for the join to produce a record.
Left Outer Join	A record from the left side (outer side) of the join is produced regardless of whether a record exists on the right side (inner side). When a record on the right side does not exist, any column from the inner side has a NULL value.
Right Outer Join	Reverse of left outer join, where the right side is the outer side and the left side is the inner side of the join.
Full Outer Join	A record is produced whether there is a match on the right side or the left side of the join.

Event Stream Processor also supports these cardinalities:

Type	Description
One-One	Keys of one side of the join are completely mapped to the keys of the other side of the join. One incoming row produces only one row as output.
One-Many	One record from the one side joins with multiple records on the many side. The one side of the join is the side where all the primary keys are mapped to the other side of the join. Whenever a record comes on the one-side of the join, it produces many rows as the output.
Many-Many	The keys of both side of the join are not completely mapped to the keys of the other side of the join. A row arriving on either side of the join has the potential to produce multiple rows as output.

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:

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

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 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.

See also

- *Creating and Modifying Simple Queries: Join* on page 43

Creating and Modifying Simple Queries: Union

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

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.

See also

- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Compute* on page 42
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Pattern* on page 47
- *Simple Queries* on page 38

Creating and Modifying Simple Queries: Pattern

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.

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, or by selecting the expression and pressing **Ctrl+F2** to open it in the pop-up editor.
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 *CCL Programmers Guide*.
6. (Optional) Use the toggle option to designate the pattern object as LOCAL or OUTPUT.

See also

- *Creating and Modifying Simple Queries: Filter* on page 40
- *Creating and Modifying Simple Queries: Aggregate* on page 41
- *Creating and Modifying Simple Queries: Compute* on page 42
- *Creating and Modifying Simple Queries: Join* on page 43
- *Creating and Modifying Simple Queries: Union* on page 46
- *Simple Queries* on page 38

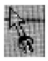
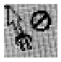
Connecting Elements

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

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

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.

Indicator	Meaning
	Connection is allowed
	Connection is not allowed

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.

See also

- *Simple Queries* on page 38
- *Shape Reference* on page 24
- *Adding a Simple Query* on page 38

Setting Key Columns

Set primary keys in the Visual editor within the Column compartment of the delta stream, window, and Flex operator shapes.

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 delta streams and windows support **PRIMARY KEY DEDUCED**. You can modify the deduced key property for these elements from the Properties view.

1. Expand the **Columns** compartment of the desired query object (delta stream, 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 stream or window.
   The column you initially selected and all other deduced key columns are now primary keys. In addition, the target stream or 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.

Editing Column Expressions for Windows, Streams, and Delta Streams

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

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.

See also

- *Column Expressions* on page 50

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.

You can specify a column name by itself or with the name of its stream or window. To specify both the column and the stream or window, use the format "stream_name.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:

Alias	Column Expression
Symbol	PriceFeed.Symbol
LastPrice	PriceFeed.Price

Alias	Column Expression
VWAP	(sum ((PriceFeed.Price * CAST (FLOAT , PriceFeed.Shares))) / CAST (float , sum (PriceFeed.Shares)))
LastTime	PriceFeed.TradeTime

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`.

See also

- *CCL Functions* on page 70
- *Operators* on page 71
- *Editing Column Expressions for Windows, Streams, and Delta Streams* on page 49

Deleting an Element

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

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.

See also

- *Adding Shapes to a Diagram* on page 28
- *Keyboard Shortcuts in the Visual Editor* on page 29

Adding Advanced Features to a Project

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

All of these advanced features are optional.

Complex Queries

Use the generic derived stream, derived window, and derived delta stream 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

Shape	Input	Output	Retains state	Preserves inserts, updates, and deletes
Derived Stream	Another stream	Stream	no	no
Derived Window	Another stream or window	Window	As defined in Keep policy (default is keep all rows)	yes **Note:** In order to derive a window from a stream, a GROUP BY clause must be included in the query.
Derived Delta Stream	Another window or delta stream	Stream	no	yes **Note:** A delta stream only accepts either inserts or deletes.

See also

- *Join Types and Restrictions* on page 44
- *Operation Codes* on page 7

- *Editing Column Expressions for Windows, Streams, and Delta Streams* on page 49

Modularity

A module in Sybase 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.

Creating a Module

Add a new module to an existing project in the Visual editor.

Create modules directly in a project when you do not plan to reuse them widely across other projects.

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

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.

See also

- *Editing a Module* on page 54
- *Creating a Module File* on page 55
- *Importing Definitions from Another CCL File* on page 55
- *Using a Module Within a Project* on page 56
- *Configuring the Loaded Module* on page 57
- *Configuring a Module Repository* on page 58

Editing a Module

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

Prerequisites

Create the module.

Task

Specific module inputs and outputs are determined by project developers. Imported modules have restrictions on editing, but you can modify module input and output nodes.

1. In the Visual editor, select the module to edit.
2. 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 Authoring perspective.
3. Click **Add Module Inputs** ().
4. In the Module Inputs dialog, select the inputs to add or remove, then click **OK**.
5. Select **Add Module Outputs** ().
6. In the Module Outputs dialog, select the outputs to add or remove, then click **OK**.
7. To access and edit the contents of the **CREATE MODULE** statement, select **Open Module Diagram** ().
8. Edit the module in the diagram that opens.
9. Add comments in the Properties view.

See also

- *Creating a Module* on page 53
- *Creating a Module File* on page 55
- *Importing Definitions from Another CCL File* on page 55
- *Using a Module Within a Project* on page 56
- *Configuring the Loaded Module* on page 57
- *Configuring a Module Repository* on page 58

Creating a Module File

Create a new, separate module file that can be imported into a project.

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.

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 *CCL Programmers Guide*.

See also

- *Creating a Module* on page 53
- *Editing a Module* on page 54
- *Importing Definitions from Another CCL File* on page 55
- *Using a Module Within a Project* on page 56
- *Configuring the Loaded Module* on page 57
- *Configuring a Module Repository* on page 58

Importing Definitions from Another CCL File

Import a module file to use the module in your project.

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.

1. Select the **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.

See also

- *Creating a Module* on page 53
- *Editing a Module* on page 54
- *Creating a Module File* on page 55
- *Using a Module Within a Project* on page 56
- *Configuring the Loaded Module* on page 57
- *Configuring a Module Repository* on page 58

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.

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.

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.

See also

- *Creating a Module* on page 53
- *Editing a Module* on page 54
- *Creating a Module File* on page 55
- *Importing Definitions from Another CCL File* on page 55
- *Configuring the Loaded Module* on page 57
- *Configuring a Module Repository* on page 58

Configuring the Loaded Module

Add or remove input and output bindings and parameter values (if any) for a specific module instance.

Active modules are created when existing module definitions are used to create new module instances.

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:

Property	Value
inputStreamOrWindow	Select the available input stream or window components from the list.
streamOrWindowInModule	Select the available stream or window to bind with existing stream or window inputs.
comment (Output only)	Add a comment or description of the output stream.
name (Output only)	Add a name to the output stream.

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** (🗂).

See also

- *Creating a Module* on page 53
- *Editing a Module* on page 54
- *Creating a Module File* on page 55
- *Importing Definitions from Another CCL File* on page 55
- *Using a Module Within a Project* on page 56
- *Configuring a Module Repository* on page 58

Configuring a Module Repository

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

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.

1. Create a new folder, or select an existing folder, to serve as the module repository.
2. In Studio, click **Edit** > **Preferences** > **Sybase Event Stream Studio** .
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**.

See also

- *Creating a Module* on page 53
- *Editing a Module* on page 54
- *Creating a Module File* on page 55
- *Importing Definitions from Another CCL File* on page 55
- *Using a Module Within a Project* on page 56
- *Configuring the Loaded Module* on page 57

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 STORE** statement. 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 a default store using the **CREATE DEFAULT 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.

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.

Task

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 7. Log Store Properties

Property	Description
File name (FILENAME)	The absolute or relative path to the folder where log store files are written. A relative path is preferred.
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.
Sync (SYNC)	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.

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

See also

- *Creating a Memory Store* on page 61

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.

Task

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 8. Memory Store Properties

Property	Description
Index Size Hint (KB) (IN-DEXSIZEHINT)	(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 chan-ces of spikes in latency. Default is 8KB.
Index Kind (INDEXTYPE)	The type of index mechanism for the stored elements. Default is **Tree**. Use **Tree** for binary trees. Binary trees are predictable in use of memory and consistent in speed. Use **Tree** for hash tables, as hash tables are faster, but they often consume more memory.

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

See also

- *Creating a Log Store* on page 59

Flex Operators

Flex operators are custom operators that let you write SPLASH 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 SPLASH.

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 SPLASH scripts.

See also

- *SPLASH* on page 11

Creating a Flex Operator in the Visual Editor

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

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 SPLASH script that is invoked when an event arrives on the associated input. In other words, these are event handlers.

   a) To edit the SPLASH script for each method, make sure the Flex shape is selected, and press **F4** to switch to the CCL editor.
   The CCL editor opens with the cursor at the CREATE FLEX statement.
   b) Edit the SPLASH script.
   c) Press **F4** 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.

See also

- *Specifying a Retention Policy* on page 35
- *Setting an Aging Policy* on page 63

Creating a Schema in the Visual Editor

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

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.
6. Optional Connect the schema to one or more streams or windows using the connector tool. Note: after selecting the connector tool, click the stream or window shape first, then the schema shape.

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".

Aging policies are an advanced, optional feature for a window or other stateful element.

1. Select **Set Aging Policy** and set values:

Value	Description
Aging Time	This is an interval value. Any record in the window that has not been updated for this much time will have the Aging Field incremented. 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 micro-seconds.
Aging Field	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.

Value	Description
(Optional) Max Aging Field Value	The maximum value that the aging field can be incremented to. If not specified, the aging field is incremented once.
(Optional) Aging Time Field	The start time of the aging process. If not specified, the internal row time is used. If specified, the field must contain a valid start time.

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

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.

Monitoring Streams for Errors

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

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

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.

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.

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.

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.

Displaying Error Stream Data

By default, error streams are LOCAL, but you can make their information available outside of the project.

In production mode, project monitoring may be performed externally. .

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 LOCAL to OUTPUT.
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 **esp_query** utility.

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.

Note: An error stream cannot monitor other error streams.

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.

Switching Between the CCL and Visual Editors

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

- To switch from the CCL editor to the Visual editor, right-click and choose **Switch to Visual** (**F4**), 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** (**F4**), or click in the main toolbar.

See also

- *Creating a Project* on page 16

- *Opening a Project* on page 18
- *Importing an Existing Project* on page 19
- *Editing a Project in the Visual Editor* on page 27

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.

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

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.

CHAPTER 4 CCL Editor Authoring

The CCL editor is a text authoring environment within ESP 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
- SPLASH 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

For CCL language details, see the *CCL Programmers Guide*.

Editing in the CCL Editor

Update and edit CCL code as text in the Studio CCL editor.

1. Click the **Authoring** tab.
2. In File Explorer, expand the project container, and double-click the `.ccl` file name to open it in the CCL editor.

   Note: Advanced CCL users can include multiple CCL files in the same project, by using an IMPORT statement to import shared schemas and module definitions from another file.
3. Begin editing text in the CCL editor window.

   Tip: If you open a `.ccl` file in the CCL editor when the same project is open in the Visual editor, the CCL editor opens in read-only mode and you cannot edit the file.

   Close both the Visual editor and CCL editor for the project, and then reopen the project in the CCL editor.

Note: Backslashes within string literals are used as escape characters. 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.

See also

- *File Explorer* on page 14
- *Switching Between the CCL and Visual Editors* on page 65
- *Compiling a Project* on page 93

CCL Editor Features

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

Table 9. CCL Editor Features

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

Keyboard Shortcuts in the CCL Editor

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

Key	Action
F3	Jump to declaration
F4	Toggle between the Visual and CCL editor
F6	Reorder CCL statements
F7	Compile

Key	Action
F11	Toggle between Authoring and Run-Test perspective
Ctrl + N	Opens new project file
Ctrl + Y	Redo
Ctrl + Z	Undo
Ctrl + Shift + L	List all keyboard shortcut assignments

Searching for Text

Find text in CCL code.

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.

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 *CCL Programmers Guide*:

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

See also

- *Simple Queries* on page 38

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.
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 ,
```

CCL Functions

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

The Sybase Event Stream Processor supports:

- Built-in functions - including aggregate, scalar and other functions
- User-defined SPLASH functions
- User-defined external functions

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

Order of Evaluation of Operations

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

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

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

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.

Operator	Meaning	Example Usage
+	Addition	3+4
-	Subtraction	7-3
*	Multiplication	3*4
/	Division	8/2
%	Modulus (Remainder)	8%3
^	Exponent	4^3
-	Change signs	-3
++	Increment Preincrement (++*argument*) value is incremented before it is passed as an argument Postincrement (*argument*++) value is passed and then incremented	++a (preincrement) a++ (postincrement)
`--`	Decrement Predecrement (--*argument*) value is decremented before it is passed as an argument Postdecrement (*argument*--) value is passed and then decremented	--a (predecrement) a-- (postdecrement)

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
```

Operator	Meaning	Example Usage
=	Equality	a0=a1
!=	Inequality	a0!=a1
<>	Inequality	a0<>a1
>	Greater than	a0!>a1
>=	Greater than or equal to	a0!>=a1
<	Less than	a0!<a1
<=	Less than or equal to	a0!<=a1
IN	Member of a list of values. If the value is in the expression list's values, then the result is TRUE.	a0 IN (a1, a2, a3)

Logical Operators

Operator	Meaning	Example Usage
AND	Returns TRUE if all expressions are TRUE, and FALSE otherwise.	(a < 10) AND (b > 12)
NOT	Returns TRUE if all expressions are FALSE, and TRUE otherwise.	NOT (a = 5)
OR	Returns TRUE if any of the expressions are TRUE, and FALSE otherwise.	(b = 8) OR (b = 6)
XOR	Returns TRUE if one expression is TRUE and the other is FALSE. Returns FALSE if both expressions are TRUE or both are FALSE.	(b = 8) XOR (a > 14)

String Operators

Operator	Meaning	Example Usage
+	Concatenates strings and returns another string. **Note:** The + operator does not support mixed datatypes (such as an integer and a string).	'go' + 'cart'

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	Syntax and Meaning	Example Usage
LIKE	Matches **WHERE** clause string expressions to strings that closely resemble each other but do not exactly match. `compare_expression LIKE pat-tern_match_expression` The LIKE operator returns a value of TRUE if **compare_expression** matches **pattern_match_expression**, 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.	Trades.StockName LIKE "%Corp%"

[] Operator

The [] operator is only supported in the context of dictionaries and vectors.

Operator	Syntax and Meaning	Example Usage
[]	Allows you to perform functions on rows other than the current row in a stream or window. `stream-or-window-name[index].column` **stream-or-window-name** is the name of a stream or window and **column** indicates a column in the stream or window. **index** is an expression that can include literals, parameters, or operators, and evaluates to an integer. This integer indicates the stream or window row, in relation to the current row or to the window's sort order.	MyNamedWindow[1].MyColumn

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

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:

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

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

```

/**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.

CHAPTER 5 Project Configurations

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

Project configuration files are created and edited separately from the project they are attached to, and are identified by their `.ccr` file extension. View and edit project configuration files in the File Explorer view in the 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. New configuration files are also created when Aleri models are converted to Event Stream Processor projects. One project may have multiple configuration files attached to it, so you can manually create new project configurations.

Creating a Project Configuration

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

1. Select **File** > **New** > **Project Configuration**.
2. Select the folder in which to store the new configuration file, and assign it a file name.
3. Click **Finish.**
   You see the CCR Project Configuration Editor window.

See also

- *Opening an Existing Project Configuration* on page 76
- *Project Configuration File Editor* on page 76
- *Advanced Project Deployment Options* on page 83

Opening an Existing Project Configuration

Open an existing project configuration file.

By default, new projects create a project configuration so each project has at least one existing project configuration.

1. Select **Window** > **Open Perspective** > **Authoring** or click the **Authoring** tab.
2. Select **Window** > **Show View** > **File Explorer**.
3. Locate the project configuration file, which appears as `<projectname>.ccr`. Double-click to open the file.

See also

- *Creating a Project Configuration* on page 75
- *Project Configuration File Editor* on page 76
- *Advanced Project Deployment Options* on page 83

Project Configuration File Editor

Using the CCR Project Configuration File Editor you can select one of five categories of information and edit in the project configuration file.

The CCR Project Configuration File Editor has five tabs, each one corresponding to one of the five categories of project configuration information.

See also

- *Creating a Project Configuration* on page 75
- *Opening an Existing Project Configuration* on page 76
- *Advanced Project Deployment Options* on page 83

Editing Cluster Parameters in Project Configuration

Configure local or remote clusters that your project can connect to for input. These clusters can then be used when configuring bindings.

1. In the CCR Project Configuration Editor window, select the **Clusters** tab.
2. Click the name of an existing cluster in the **All Clusters** pane to edit that cluster's information or click **Add** to add a new cluster .
   The editor displays the **Cluster Details** pane.
3. Enter the requested information in the **Cluster Details** pane.

Field	Description
Name	Enter the hostname of the cluster.
Type	Toggle between local (no server information necessary) and remote (server information must be known) cluster connection options.
User Name	Enter a user name to use when logging in to the cluster.
Password	Enter a password to use when logging in to the cluster.

4. (Optional) Click **Encrypt** after entering the user name or password.
   a) Fill in the required fields in the **Content Encryption** pane, including **Cluster URI** (comprised of your host name and port number (`<HOST>:<PORT>`) and credential fields.
   b) Click **Encrypt**.
   The editor redisplays **Cluster Details** pane with the field you chose to encrypt (either the user name or password) filled with randomized encryption characters.

   Note: To reset the encryption, click **Encrypt** beside the appropriate field and click **Reset** when the**Already Encrypted** pop-up is displayed.
5. To add a master cluster and children cluster nodes:
   a) In Cluster Details, select **remote** as the type.
   b) Right-click the cluster and select **New** > **Cluster Manager**.
   c) Configure each cluster node by selecting it and adding host and port information in the Cluster Manager field in the Cluster Manager Details pane.

See also

- *Editing Bindings in Project Configuration* on page 77
- *Editing Adapter Property Sets in Project Configuration* on page 79
- *Setting Parameters in Project Configuration* on page 80
- *Editing Advanced Options in Project Configuration* on page 80

Editing Bindings in Project Configuration

Configure bindings between input streams or windows in a project to output streams or windows in other projects.

Prerequisites

You must have verified that the streams or windows you want to bind have:

- Compatible schema.
- The same datatype for each field name.
- The same column order.

- The same number of columns.

Task

Projects can be bound to other projects, allowing one project's input stream or window to receive its input from the output stream or window of another project. Binding projects is similar to attaching an input adapter to an input stream or window, but is more efficient as it directly connects the output of one project to the input of the other. Stream binding is only supported from the receiving project. You cannot initiate a binding from the publishing side.

Bindings can be local, within the same cluster, or can connect a project in one cluster to a project in a different cluster. Binding information is specified in the project configuration (CCR) file so that binding references may be changed at runtime, allowing the project to be used in multiple environments.

1. In the CCR Project Configuration editor, select the **Bindings** tab.
2. To add a binding, click **Add**, or to display a list of available streams/windows, click **Discover**.
3. To configure individual binding items, use the **Binding Details** pane on the right side of the CCR Project Configuration editor.

Field	Description
Binding name	(Optional) Apply a name to the binding.
Local stream/window	Enter the local stream/window information (for example, localStream1) or click **Discover** to view and select from a list of running streams/ windows.
Cluster	Select the cluster to bind to. **Note:** You must have previously defined one or more clusters in the Run-Test perspective and added the cluster of interest in the Cluster tab.
Workspace	Enter the workspace data (for example, ws1) or click **Discover** to view and select from a list of running workspaces.
Project	Enter the project to access (for example, project1) or click **Discover** to view and select from a list of running projects.
Remote stream/win-dow	Enter the remote stream/window information (for example, remote-Stream1) or click **Discover** to view and select from a list of running streams/windows.

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

See also

- *Editing Cluster Parameters in Project Configuration* on page 76
- *Editing Adapter Property Sets in Project Configuration* on page 79
- *Setting Parameters in Project Configuration* on page 80

- *Editing Advanced Options in Project Configuration* on page 80

Editing Adapter Property Sets in Project Configuration

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

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

1. In the CCR Project Configuration editor, select the **Adapter Properties** tab.
2. To create a new adapter property node, click **Add**.
3. In the Property Set Details pane, define a name for the property node.
4. To add a new property to a property set, right-click the set and select **New** > **Property**.

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

5. To configure a property:
   a) In the Property Details pane, define a name for the property.
   b) Enter a value for the property.
6. (Optional) To encrypt the property value:
   a) Select the property value and click **Encrypt**.
   b) Enter the required fields, including Cluster URI and credential fields.
   c) Click **Encrypt**.
   The value, and related fields are filled with randomized encryption characters.

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

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

See also

- *Editing Cluster Parameters in Project Configuration* on page 76
- *Editing Bindings in Project Configuration* on page 77
- *Setting Parameters in Project Configuration* on page 80
- *Editing Advanced Options in Project Configuration* on page 80

Setting Parameters in Project Configuration

Edit parameter definitions and remove deleted parameters.

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

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

   Note: You cannot modify the parameter **Name** field.
3. To remove deleted parameter definitions from the list, select **Remove**, which is located at the top of the list.

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

See also

- *Editing Cluster Parameters in Project Configuration* on page 76
- *Editing Bindings in Project Configuration* on page 77
- *Editing Adapter Property Sets in Project Configuration* on page 79
- *Editing Advanced Options in Project Configuration* on page 80

Editing Advanced Options in Project Configuration

Modify project deployment properties, project options, and instances in a project configuration file.

1. In the CCR Project Configuration editor, select the **Advanced** tab.
2. If no project deployment item exists, select **Add**.
3. Choose a project deployment type from the Project Deployment Details window. The options are:

Type	Description
Non-HA	Non-HA deployments create one project option item and one instance item as children under the project deployment item.
HA	HA deployments create one project option item and two instance items as children under the project deployment item. HA provides for hot project fail-over between instances.

4. To add an option, right-click the project options item and select **New** > **option**.

   This table outlines all available project options that can be set using the Project Configuration view in ESP Studio:

Project Option	Description
on-error-discard-record	If set to true, the record being computed is discarded when a computation failure occurs. If set to false, any uncomputed columns are null-padded and record processing continues. The default value is true. **Note:** If the computation of a key column fails, the record will be discarded regardless of this option.
on-error-log	If set to true, any computation errors that occur will be logged in the error message. The default value is true.
java-classpath	Set the java classpath. Value is a filepath to the classpath file.
java-max-heap	Set the max java heap for the project. Default value is 256 megabytes.
utf8	Enable UTF-8 functionality on the server (by default, this is feature is off). Default value is false, set to true to enable.
precision	Set decimal display characteristics for number characters in the project. Default value is 6.
command-port	Set the command port number. This advanced option should not generally be set. If the port is 0, or out of range 1-65535, the program selects an arbitrary port. To define a specific port, set a value between 1 and 65535.
sql-port	Set the SQL port number. This advanced option should not generally be set. If the port is 0, or out of range 1-65535, the program selects an arbitrary port. To define a specific port, set a value between 1 and 65535.
gateway-port	Set the gateway port number. This advanced option should not generally be set. If the port is 0, or out of range 1-65535, the program selects an arbitrary port. To define a specific port, set a value between 1 and 65535.
time-granularity	Define time granularity within the project. This option specifies, in seconds, how often the set of performance records—one per stream and one per gateway connection—is obtained from the running Event Stream Processor. By default, time granularity is set to 5. Set this option to 0 to disable monitoring; this also optimizes performance.

Project Option	Description
debug-level	Set a logging level for debugging the project, ranging from 0-7. Where each number is represents the following: • 0: LOG_EMERG - system is unusable • 1: LOG_ALERT - action must be taken immediately • 2: LOG_CRIT - critical conditions • 3: LOG_ERR - error conditions • 4: LOG_WARNING - warning conditions • 5: LOG_NORMAL - normal but significant conditions • 6: LOG_INFO - informational • 7: LOG_DEBUG - debug level messages **Note:** When changing settings, stop and remove the project from the Server, then redeploy the project.
bad-record-file	To save bad records to a file, select the bad-record-file option for **Project Type**, and indicate the file name of an ESP project for **Value Field**. If a file name is not specified, bad records are discarded.
memory	Set memory usage limits for the project. Default is 0, meaning unlimited.
optimize	Suppresses redundant store updates. Default value is false, set to true to enable.
ignore-config-topology	Enable this to ignore topology between projects. Default is false, set to true to enable.
time-interval	Set the constant interval expression that specifies the maximum age of rows in a window. By default, in seconds, set to 0, meaning no timer.

Note: Each project option can only be added once. Implemented project options are no longer available in the drop-down list.

5. To configure an option item, complete these fields:

Option	Description
Name	Select from the list of available options shown in the above table.
Value	Enter a value for the property option.

6. To add an affinity under the instance item, right-click the instance item and select **New** > **affinity**. Complete these fields:

Option	Description
Name	Enter a name for the affinity item.

Option	Description
Strength	Select a strength level.
Type	Select a type. (for example, **controller**).
Charge	Select a charge.

7. To remove items from the All Advanced Configurations list:

- Select a project deployment item and click **Remove**.
- Right-click an option or affinity item and select **Delete**.

See also

- *Editing Cluster Parameters in Project Configuration* on page 76
- *Editing Bindings in Project Configuration* on page 77
- *Editing Adapter Property Sets in Project Configuration* on page 79
- *Setting Parameters in Project Configuration* on page 80
- *Advanced Project Deployment Options* on page 83

Advanced Project Deployment Options

Project deployment options determine how your project is deployed in a cluster and how it functions at runtime. These parameters, including project options, active-active instances, failover intervals, and project deployment type options, are set in the CCR file manually or within Studio.

Active-Active Deployments

Active-active deployments are available only when you define the project as an `ha-project` in the CCR file. An active-active deployment means that two instances of a project run simultaneously in a cluster. The two instances of the project are started by the cluster manager on two different hosts.

One instance of the project is elected as 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

The number of instances available depends on the deployment type chosen by the user, either high availability (HA) or Non-HA. When a project is configured in HA (active-active) mode, two instances are created: primary and secondary. You can set affinity and cold failover options for each instance, including failover intervals and failure per interval options.

Failover

A project fails when it does not run properly or stops running properly. A failover occurs when a failed project or server switches to another server to continue processing. Failovers may result in a project restart, if defined. Restarts can be limited based on definition of failure intervals and restarts per interval. Failover options, accessed using an instance configuration, include:

Field	Description
Failover	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.
Failures per interval	Specifies the number of restarts the project can attempt within a given interval. This count can be reset to zero by a manual start of the project or if failures are dropped from the list because they are older than the size of the interval.
Failure interval	(Optional) This specifies the time, in seconds, that make up an interval. If left blank, the interval time is infinite.

Affinities

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

- Controller – Used for Active-Active and non Active-Active configurations. You can have more than one affinity for each controller, but there can only be one strong positive controller affinity.
- Instance – Used only for Active-Active configuration, an instance creates two affinities that can apply to each separate project server.

These parameters must be defined for each affinity:

Field	Description
Name	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 should refer to the second instance.
Strength	**Strong** or **weak**. Strong requires the project to run on a specific controller, and no others. If weak, the project preferentially starts on the defined controller, but if that controller is unavailable, it may start on another available controller.

Field	Description
Charge	**Positive** or **negative**. If positive, the project runs on the controller. If negative, the project does not run on the controller.

See also

- *Creating a Project Configuration* on page 75
- *Opening an Existing Project Configuration* on page 76
- *Project Configuration File Editor* on page 76
- *Editing Advanced Options in Project Configuration* on page 80

CHAPTER 6 Running Projects in Studio

In Studio, projects can be run on either a local or a remote cluster, using any of three methods of authentication, and multiple projects can be run simultaneously on different clusters and in separate workspaces.

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 users to work with multiple projects simultaneously.

A local cluster allows users to work on projects from their local machine. Internet access is not required. By default, clicking **Run Project** runs the project on the local cluster. If the local cluster is not running, it is started automatically.

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 ESP Studio. Once the cluster is running, you can connect to it from Studio and run the project.

Connecting to the Local Cluster

Connect ESP Studio to the local cluster and run the project there.

Run Project ▶ enables you to run projects on the local cluster from either the **Authoring** perspective or the **Run-Test** perspective.

1. In the **Authoring** perspective.
   a) Select a project and open it in either the Visual Editor or the CCL Editor.
   b) Select **Run Project** ▶.
   c) You are prompted to provide the required user name and password. Use the default user name "studio" and enter any password.

   Note: The password you use is stored in memory and is valid for your entire Studio session. If you forget your password, shut down and restart Studio.

   The Server View in the 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 **Run-Test** perspective.
   a) Select **Run Project** .
   b) You are prompted to provide the required user name and password. Use the default user name "studio" and enter any password.
   The system displays a list of projects in the **Select Project** pop-up window.
   c) Select the project that you want to run.

   Note: If you already have a project running, you need to select the drop-down menu to the immediate right of **Run Project** to bring up the list of projects.

ESP Studio acts as a node (cluster manager): automatically connecting to the local cluster and running the project on it.

Connecting to a Remote Cluster

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

Prerequisites

The remote cluster connection must be configured in Studio and the remote cluster's administrator must have started the remote cluster outside of ESP Studio. If using Kerberos authentication, run a program outside of ESP Studio to obtain a current Ticket Granting Ticket (TGT).

Task

1. Select the **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**

   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 remote 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.

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.

Task

1. In the Server View, select **Studio Preferences > Sybase Event Stream Processor Studio > Run Test Preferences > Security Settings**.
   Studio displays the **Security Settings** screen.
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.

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.

Task

1. In the Server View, select **Studio Preferences** > **Sybase Event Stream Processor Studio** > **Run Test Preferences** > **Security Settings**.
   Studio displays the **Security Settings** screen.
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.

Configuring a Remote Cluster Connection

Use 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.

Task

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 **Sybase Event Stream Processor Studio** > **Run Test Preferences**. Select **New**.

   Studio displays the **New Server** screen.
2. Enter the host name and port number, separated by a colon, to use when connecting to the remote cluster. For example, `myserver.mycompany.com: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**.

In the 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.

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

Modifying a Remote Cluster Connection

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

If the administrator of the remote cluster changes the authentication settings of the remote cluster you must modify the remote cluster connection in Studio accordingly.

1. In the Server View, select **Studio Preferences > Sybase Event Stream Processor Studio > Run Test Preferences**.
   Studio displays the **Run Test Preferences** screen.
2. Select an existing server connection.
   The **Remove** and **Edit** buttons are activated.
3. Click **Edit**.
   Studio displays the **Remote Server Connection** screen.
4. Make your changes and click **OK**.
   Studio displays the **Run Test Preferences** screen.
5. Click **OK** to save your changes.

CHAPTER 7 Running and Testing a Project

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

Starting the Run-Test Perspective

Access the Run-Test perspective for toolbars and views that simplify testing, monitoring, debugging, and examining Event Stream Processor projects.
Click the **Run-Test** tab at the top of the Studio main window to see the Run-Test perspective.

If the Run-Test tab is not visible, from the main menu select **Window > Open Perspective > Run-Test**.

Compiling a Project

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

1. (Optional) Set CCL compiler options.
   a) Choose **Edit > Preferences**.
   b) Expand the tree view to **Sybase Event Stream Processor > Run Test Preferences > Compiler Options**.
   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 Authoring perspective, in File 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. If you want 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. If you want to compile and run the project, click **Run Project** .

   The project automatically compiles and runs. The Server View in the 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.

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.

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.

Viewing Problems

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

Prerequisites

Open the Authoring Perspective.

Task

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:

Option	Action
Go to	Highlight the problem in the .ccl file. The CCL editor opens in read-only mode.
Copy	Copy error details to the clipboard. When you exit Studio, the contents of problems view are removed. Use this option to save off errors.

Option	Action
Show in	Display details in Properties view.
Quick Fix	(Disabled)
Properties	Display details in a dialog box.

4. (Optional) Click the View menu dropdown to see more options.
5. Click the **Console** tab to view compiler results.

Running a Project

Running a project automatically starts the project either on a local cluster or on another connected cluster.

Prerequisites

To run a project in a workspace other than the default, ensure that one or more connected workspaces are available.

Task

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 Authoring or the 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 Project in Workspace**. Then select the workspace where this project will run.

   The project runs and shows results in Run-Test perspective.

Server View

The Server View shows 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 *Administrators Guide*.

See also

- *Chapter 6, Running Projects in Studio* on page 87
- *Performance Monitor* on page 99
- *Event Tracer View* on page 104

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.

1. In the 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** . If available, the publish date of the stream appears.
4. (Optional) To save data from the Stream View, click **Clipboard Copy** .

Controlling the Pulse Rate for Viewing 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.

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

1. Choose **Edit** > **Preferences**.
2. In the left pane, expand **Sybase Event Stream Processor Studio**, and then expand **Run Test Preferences**.
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.

Uploading Data to ESP 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, either on a local or remote cluster.

Task

1. In the 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: ESP Server supports ESP binary (`.bin`), ESP XML (`.xml`), and comma-separated values and text (`.csv` or `.txt`) files. Refer to the *Supported File Formats* topic for details.
6. Click **Upload**. A progress bar tracks the upload status.

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

UI control	Action
Remove File	Discard a previously selected file from the Input File(s) menu.
Cancel	Cancel a file upload currently in progress. **Note:** Any data sent before the upload is cancelled is still processed.

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

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.

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

1. In the 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.

See also

- *Manual Input Settings* on page 112

Activating a Project

Start monitoring the selected project using one or more input views: Playback, File Upload, SQL Query, Monitor, and Event Tracer.

1. Open the Studio in the Run-Test Perpsective.
2. In the **Activate Project** pane, specify the views you want.
   a) Click **None** to clear any existing selections.

b) Click **All** to select all of the views or click the checkbox next to those views you want to start up.

3. If you already have a view open showing another project, click **Override active view** to stop monitoring that project and monitor the project you are activating instead.

   If you do not check this option, any views currently monitoring another project will continue to do so.

4. Select the project you want to activate and click **OK**.
   Studio displays the name of the project you activated in the upper left hand corner and starts monitoring the project in each of the selected views.

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.

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.

Task

1. In the 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.

See also

- *Saving a Performance Diagram as an Image* on page 100

Saving a Performance Diagram as an Image

Save a performance diagram.

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

1. In the 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.

See also

- *Running the Monitor* on page 99

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.

1. In the 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.

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 10. Playback View Options

Feature	Description
Select playback file	Select file format to use with Event Stream Processor recorder.
Start playback	Starts playing the current playback file.
Stop playback	Stops playback or record, closes the associated file and closes the associated playback or record context.
Start Recording	Prompts user to select the file in which to store recorded data and starts Event Stream Processor recorder
At timestamp rate	This slider is used during playback to vary the rate of playback

Table 11. Playback Mode Options

Feature	Description
Full rate	**Full rate** indicates that the speed of playback is not imposed by ESP Studio. **Full rate** is dependent on factors such as the computer that is running ESP Studio, or network latency.
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. If you check **Use Recorded Time**, the playback file runs as if it is the time when the data was recorded. Otherwise, the playback file uses the current time and plays as if produced now.

Feature	Description
Rec/ms	The records-per-millisecond (**rec/ms**) mode lets playback occur at a records per millisecond rate. The option allows you to set an initial rec/ms rate that you can then modify using the **At timestamp rate** slider tool.

The ESP Studio Recorder supports these file formats:

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

Refer to the *Supported File Formats* topic for details.

Note: Binary files recorded in previous releases cannot be played back unless they are first converted to the new binary format using **esp_convert**. See *Publish and Subscribe Executables > esp_convert* for information on how to to convert binary files.

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 an ESP server running in a little-endian machine, and vice-versa.

Recording Incoming Data in a Playback File

Record data from in-flowing data to Event Stream Processor to a playback file that you can save and view at a later time.

Prerequisites

You must be connected to ESP Server, and your streams and windows must be visible in the Stream Viewer.

Task

In the Playback view:

1. Click **Select Project** (🗂).
2. Select the project you want to record.
3. 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.
6. Click **OK**.
7. Select or create a file in which to save the recording.
8. Click **OK**.
9. 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.
10. Click **Stop** to stop recording.

See also

- *Playing Recorded Data* on page 103

Playing Recorded Data

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

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

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.

    Note: You can delete an item from the history using the either the **Remove** button or **Delete** key. Modifications to the playback history are permanent.
3. Click **Play** to begin playback.
    The data appears in the Stream Viewer, by default, at the rate it was recorded.

See also

- *Recording Incoming Data in a Playback File* on page 102

Debugging

The 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 *Utilities Guide*.

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 12. Event Tracer View

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

See also

- *Debugging with Breakpoints and Watch Variables* on page 106

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.

Task

1. In the Run-Test Perspective, either:

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

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

2. Double-click a node to show the corresponding stream's data in the Console view.
3. 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**.

The shapes in the Event Tracer view change color.

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.

1. In the 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.

Debugging with Breakpoints and Watch Variables

ESP 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 13. Studio Breakpoint Buttons

Button	Function
Trace On	Instructs Event Stream Processor to begin tracing (debugging). This parameter must be set to use the Event Stream Processor breakpoint APIs.
Trace Off	Stops tracing (debugging).
Step Project	Steps the running Event Stream Processor.
Pause Project	Pauses the running Event Stream Processor.
Enable All Breakpoints	Enables all breakpoints in the list.
Disable All Breakpoints	Disables all breakpoints in the list.
Insert Breakpoint	Inserts a breakpoint item into the watch table.
Insert Watch	Inserts a watch item into the watch table.
Print Breakpoint Data to Console	Prints the breakpoint and pause state data for the current Event Stream Processor to the console.

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

Table 14. Breakpoint Commands

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

Note: Breakpoints and watch variables are persisted to the workspace.

See also

- *Event Tracer View* on page 104

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).

See also

- *Adding Breakpoints* on page 107
- *Watch Variables* on page 108
- *Adding Watch Variables* on page 109
- *Pausing the Event Stream Processor* on page 109
- *Stepping the Event Stream Processor* on page 110

Adding Breakpoints

Add breakpoints to Event Stream Processor.

Prerequisites

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

Task

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

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

The breakpoint appears in the Debugger view within the Breakpoint table.

8. 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.

See also

- *Breakpoints* on page 107
- *Watch Variables* on page 108
- *Adding Watch Variables* on page 109
- *Pausing the Event Stream Processor* on page 109
- *Stepping the Event Stream Processor* on page 110

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
- Variable of a Flex 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.

See also

- *Breakpoints* on page 107
- *Adding Breakpoints* on page 107
- *Adding Watch Variables* on page 109

- *Pausing the Event Stream Processor* on page 109
- *Stepping the Event Stream Processor* on page 110

Adding Watch Variables

Add a watch element to a breakpoint.

Prerequisites

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

Task

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.

See also

- *Breakpoints* on page 107
- *Adding Breakpoints* on page 107
- *Watch Variables* on page 108
- *Pausing the Event Stream Processor* on page 109
- *Stepping the Event Stream Processor* on page 110

Pausing the Event Stream Processor

Pause Event Stream Processor.

Prerequisites

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

Task

1. In the Debugger, click **Pause Project** ( ).
2. To resume Event Stream Processor, click **Trace Off** followed by **Trace On**.

See also

- *Breakpoints* on page 107
- *Adding Breakpoints* on page 107
- *Watch Variables* on page 108
- *Adding Watch Variables* on page 109
- *Stepping the Event Stream Processor* on page 110

Stepping the Event Stream Processor

Single-step Event Stream Processor.

Prerequisites

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

Task

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.

See also

- *Breakpoints* on page 107
- *Adding Breakpoints* on page 107
- *Watch Variables* on page 108
- *Adding Watch Variables* on page 109
- *Pausing the Event Stream Processor* on page 109

CHAPTER 8

Customizing the Studio Work Environment

Customize your Studio interface to work the way you prefer.

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

Editing Studio Preferences

Edit preferences to customize the Studio environment.

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

1. Choose **Edit > Preferences**.
2. Expand **Sybase Event Stream Processor Studio**, 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 Preferences –** Set defaults for server connections, add new connections, set limits and filters for the StreamViewer and Server view, and set other options for running projects in Studio.
   - **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.
   - **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.

See also

- *Changing the Display of Diagrams* on page 30

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.

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

Setting	Description
timestamp	Indicates the default value for timestamp types. Click **Select** to open a calendar dialog and choose a default timestamp with millisecond precision.
money(n)	Indicates the default value for money types of varying precision, where n represents the number of places allowed after the decimal point. Set default values for money types with up to 15 points of precision.

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.

See also

- *Manually Entering Data to a Stream* on page 98

Rearranging Views in a Perspective

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

1. Click in 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 drop cursor icon changes appearance to help you determine where the view can be docked.

Table 15. Drop cursors

Drop cursor	Cursor name	Description
⬆	Dock Above	Dock above the view that is under the cursor.
⬇	Dock Below	Dock below the view that is under the cursor.
➡	Dock to the Right	Dock to the right of the view under the cursor.
⬅	Dock to the Left	Dock to the left of the view under the cursor.
🗗	Stack	The view appears as a tab in the view under the cursor.

Drop cursor	Cursor name	Description
🛇	Restricted	The view cannot be docked. For example, a view cannot be docked in an editor.

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.

Moving the Perspective Shortcut Bar

The Perspective shortcut bar runs horizontally in the upper left corner of a perspective by default.

The Perspective shortcut bar can be docked horizontally at the top right, or vertically to the left of a perspective.

1. Right-click in the Perspective shortcut bar to open its context menu.
2. Do one of the following:

Select	To dock the shortcut bar
Dock on > Top Right	At the top right, horizontally adjacent to the main toolbar.
Dock on > Top Left	At the top left, horizontally below the main toolbar. This is the default.
Dock on > Left	At the top right, vertically on the side of a perspective.

APPENDIX A Adapter Support for Schema Discovery

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

Adapter	Supports Schema Discovery	Properties
Adaptive Server Enterprise Output	Yes	DB Service Name The name of the database service that represents the ASE database into which information will be loaded.
AtomReader Input	No	—
Database Input	Yes	Database Service Name of database service from which the adapter obtains the database connection.
Database Output	Yes	Database Service Name of service entry to use.
Excel Add-In	No	—
File CSV Input	Yes	Directory The absolute path to the data files you want the adapter to read.
File CSV Output	No	—
File FIX Input	No	—
File FIX Output	No	—
File XML Input	Yes	Directory The absolute path to the data files you want the adapter to read.
File XML Output	No	—
FIX Input	No	—

Adapter	Supports Schema Discovery	Properties
FIX Output	No	—
Flex Output	No	—
HTTP Input	No	—
JMS CSV Input	Yes	• Delimiter – field delimiter • Connection Factory – connection factory class name • JNDI Context Factory – context factory for JNDI context initialization • JNDI URL • Destination Type • Destination Name
JMS CSV Output	No	—
JMS Custom Input	No	—
JMS Custom Output	No	—
JMS FIX Input	No	—
JMS FIX Output	No	—
JMS Object Array Input	Yes	• Connection Factory – connection factory class name • JNDI Context Factory – context factory for JNDI context initialization • JNDI URL • Destination Type • Destination Name
JMS Object Array Output	No	—
JMS XML Input	Yes	• Connection Factory – connection factory class name. • JNDI Context Factory – context factory for JNDI context initialization • JNDI URL • Destination Type • Destination Name

Adapter	Supports Schema Discovery	Properties
JMS XML Output	No	—
Kdb Input	Yes	• KDB Server • KDB Port • KDB User • KDB Password
Kdb Output	Yes	• KDB Server • KDB Port • KDB User • KDB Password
Log File Input	No	—
Random Tuples Generator Input	No	—
Replication Server Input	Yes	• RSSD Host • RSSD Port • RSSD Database Name • RSSD User Name • RSSD Password
Reuters Marketfeed Input	Yes	Discovery Path
Reuters Marketfeed Output	No	—
Reuters OMM Input	Yes	Discovery Path
Reuters OMM Output	No	—
RTView Output	No	—
Sample Input Adapter	Yes	Discovery Directory Path
Sample Output Adapter	Yes	Discovery Directory Path
SMTP Output	No	—
Socket (as Client) CSV Input	No	—

Adapter	Supports Schema Discovery	Properties
Socket (as Client) CSV Output	No	—
Socket (as Client) XML Input	No	—
Socket (as Client) XML Output	No	—
Socket (as Server) XML Input	No	—
Socket (as Server) XML Output	No	—
Socket (as Server) CSV Input	No	—
Socket (as Server) CSV Output	No	—
Socket FIX Input	No	—
Socket FIX Output	No	—
Sybase IQ Output	Yes	DB Service Name The name of the database service that represents the IQ database into which information will be loaded.
Open Input and Output	No	—
Tibco Rendezvous Input	No	—
Tibco Rendezvous Output	No	—
NYSE Input	Yes	Discovery Directory Path Absolute path to the adapter discovery directory.

See also

- *Schema Discovery* on page 32
- *Discovering a Schema* on page 33

Index