



Getting Started Guide

Sybase Event Stream Processor

5.0

DOCUMENT ID: DC01622-01-0500-03

LAST REVISED: May 2012

Copyright © 2012 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

CHAPTER 1: Introduction to Sybase Event Stream Processor	1
Event Streams	2
Event Stream Processor Compared to Databases	2
Data-Flow Programming	3
ESP Projects: Adapters, Streams, Windows, and Continuous Queries	3
Streams Versus Windows	4
Schemas	5
Inserts, Updates, and Deletes	5
Product Components	6
Input and Output Adapters	7
Custom Adapters	8
Authoring Methods	8
Continuous Computation Language	9
SPLASH	9
CHAPTER 2: Exploring a Project in ESP Studio	11
Studio Workspace Basics	11
Examples	12
Loading a Sample Project	13
CHAPTER 3: Visual Editor Authoring	15
Diagrams	15
Visual Authoring Environment	16
Editing a Project in the Visual Editor	17
Adding Shapes to a Diagram	18
Connecting Elements	18

Changing the Display of Diagrams	19
Building a Simple Project	19
Sample Project Diagram	19
Creating the Sample Project	20
Adding an Input Adapter	21
Schema Discovery	22
Discovering a Schema	23
Adding an Input Window Manually	24
Adding Simple Continuous Queries	25
Simple Queries	26
Creating an Aggregate as a Simple Query	27
Creating a Join as a Simple Query	28
Completing the Sample Project	29
Deleting an Element	30
CHAPTER 4: CCL Authoring	31
Editing in the CCL Editor	31
CCL for the Sample Project	32
CHAPTER 5: Project Execution and Testing	33
Run-Test Perspective	33
Server View	34
Connecting to a Local Cluster	34
Viewing the Data in a Stream or Window	35
Other Tools for Running and Testing Projects	35
Index	37

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:

CHAPTER 1: Introduction to Sybase 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

Event Streams

An business event is a message that contains information about an actual business event that occurred. Many business systems produce events when 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
- 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
- Transaction events, which occur each time a record is added to a database or updated in a database

Many applications are already designed to produce events in real time, typically publishing them on a message bus. Applications that are not designed in this way can be “event enabled” using tools such as Sybase® Replication Server®, which can monitor transaction logs to produce a real-time stream of events based on application database updates .

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 is not a replacement for databases. Databases excel at storing and querying static data, and reliably processing transactions. However, databases are not effective at continuously analyzing fast moving streams of data.

- 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 there are many queries that must be continuously evaluated.
- Databases do not use incremental processing. Event Stream Processor can evaluate queries incrementally as data arrives.

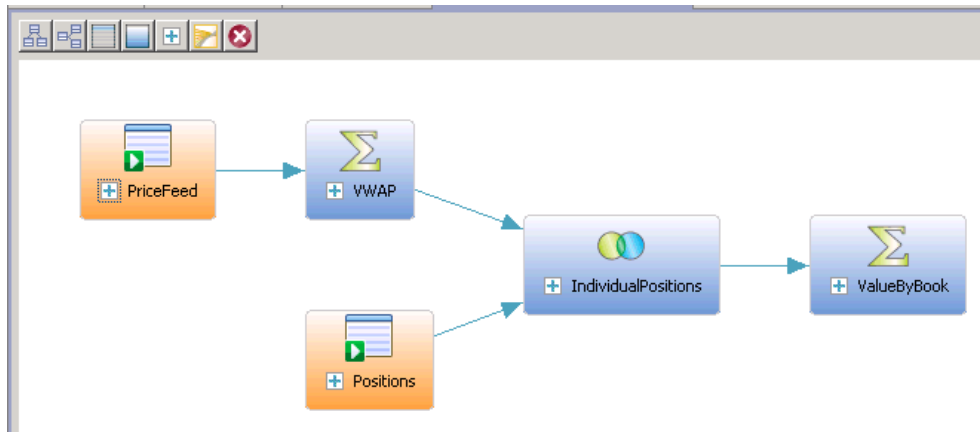
Event Stream Processor is not an in-memory database. It shares some traits with in-memory databases in that it operates in and holds all data in memory, to achieve desired speed. However, unlike an in-memory database, that is designed to efficiently process on-demand queries, Event Stream Processor uses a data-flow architecture that is optimized for continuous event-driven queries.

Data-Flow Programming

In data-flow programming, you define a set of event streams and the connections between them, and apply operations to the data as it flows from 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 runs on a separate thread and processes an event only when it is received from another operation. No other coordination is needed between operations.

Figure 1: Data-Flow Programming



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

An ESP project defines 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.
- **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.
- **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.

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 using the `esp_query` tool to query output windows in a running ESP project. For more information see the *Utilities Guide*.

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.

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.

See also

- *Adding an Input Window Manually* on page 24
- *Adding Simple Continuous Queries* on page 25

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).

See also

- *Schema Discovery* on page 22
- *Discovering a Schema* on page 23

Inserts, Updates, and Deletes

Operation Codes (opcodes) associate insert, update, and delete events with a window. They simplify development and improve performance by applying these events automatically.

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 insert, update, and delete operations associated with incoming events. You can tag events with an opcode, a special field that indicates whether the event is an insert event, an update event, or a delete event. There is also an upsert opcode, which either updates an existing record with a matching key, or inserts a new record.

Input windows apply insert, update, and delete events to the data in the window directly, as events arrive. Inserts, updates, and deletes are propagated through the query graph, that is, all downstream derived windows. Thus, when an event updates or deletes a record in an input window, it automatically applies to any downstream derived windows. This native handling of updates and deletes provides high performance and simplicity. Users do not need to manually define the logic to examine incoming events and determine how to apply them to a window.

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.

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.

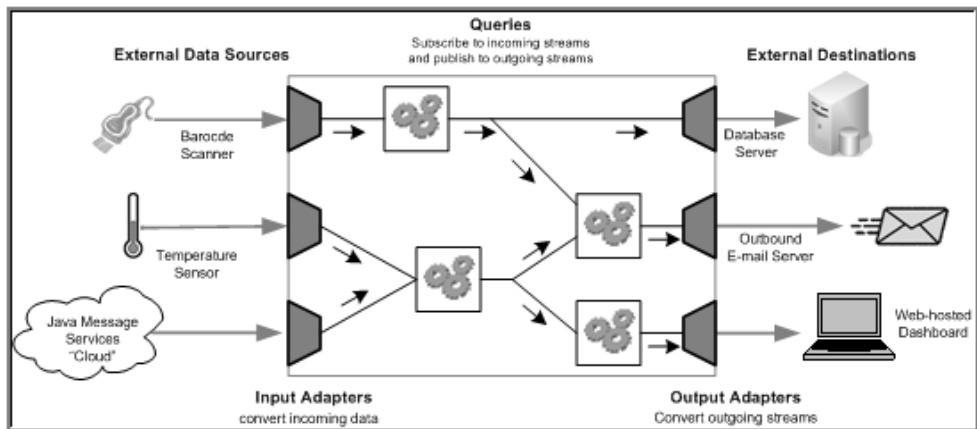
External sources or destinations can include:

- Data feeds
- Sensor devices
- Messaging systems
- Radio frequency identification (RFID) readers
- E-mail servers
- Relational databases

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.

The following illustration shows a series of input adapters that translate messages from a temperature sensor, bar code scanner, and a Java Message Service (JMS) cloud into formats compatible with Event Stream Processor. After the data is processed using various queries within Event Stream Processor, output adapters convert the result rows into updates that are sent to an external database server, e-mail server, and Web services dashboard.

Figure 2: Adapters in Event Stream Processor



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

See also

- *Adding an Input Adapter* on page 21

Custom Adapters

In addition to the adapters provided by Event Stream Processor, you can write your own adapters to integrate into the server. You can design adapters to handle a variety of external requirements that the standard adapters cannot manage.

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.

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 variety of simple 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, issue commands to the server, 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 data 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. 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.

Exploring a Project in ESP Studio

Use the sample projects in ESP Studio to learn about project structure and how to navigate perspectives and views.

Start by exploring Studio and learning what you can do in each perspective. Then, use the sample projects to see examples of different project structures and diagrams in the Visual editor.

Note: For more information on tasks and concepts introduced in the *Getting Started Guide*, see the *Studio Users Guide*.

1. Double-click the **Sybase ESP Studio** shortcut on the desktop to start the Studio.
2. (Optional) On the Welcome screen, use the buttons to navigate, or close the Welcome screen tab.
 - Click **Product Overview** or **Getting Started** to open the help.
 - Click **Learning** to open Studio with the Learning perspective active.
 - Click **Studio** to open with the Authoring perspective active.

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 1. User Activities in Studio Perspectives

Perspective	Activities
Authoring	<ul style="list-style-type: none"> • 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

Perspective	Activities
Learning	<ul style="list-style-type: none"> • 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 <hr/> <p>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.</p>
Run-Test	<ul style="list-style-type: none"> • 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

Examples

Event Stream Processor includes several example projects.

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

Also see the sample CCL and SPLASH code in the *Examples Guide*, the *CCL Programmers Guide*, and the *SPLASH Programmers Guide*.

See also

- *Sample Project Diagram* on page 19
- *Building a Simple Project* on page 19

Loading a Sample Project

Load one of the example projects installed with the product, so that you can view it in your workspace.

1. Navigate to the Learning perspective.
2. In Examples view, click one of the **LOAD** buttons to load the example into your workspace.
3. On the Start Example Project dialog, click **Cancel**.

For now, you only want to get the project into your workspace, so that you can view it in the Visual editor. Later you can run the project.

The example project opens in Authoring perspective, with the diagram (.cclnotation file) open in the Visual editor.

4. Explore the diagram by clicking the shapes, and clicking the plus sign to open compartments to see column or property details.

Note: Do not edit the example, and especially do not save any changes to the examples.

5. (Optional) To see the corresponding CCL code, right-click and choose **Switch to Text (F4)**.
The diagram closes, and the project opens in the CCL editor.
6. Close the project without saving any changes.

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 displays 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 project can have multiple diagrams. You cannot share a diagram among multiple projects.

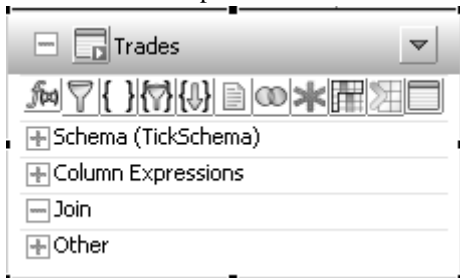
When you add a shape or other element to a diagram, it is automatically added to the project when you save. You can delete an element from a diagram only, or from the project.

Display diagrams in verbose or iconic mode:

- **iconic** – compartments are collapsed to save space.



- **verbose** – all compartments in elements are visible.



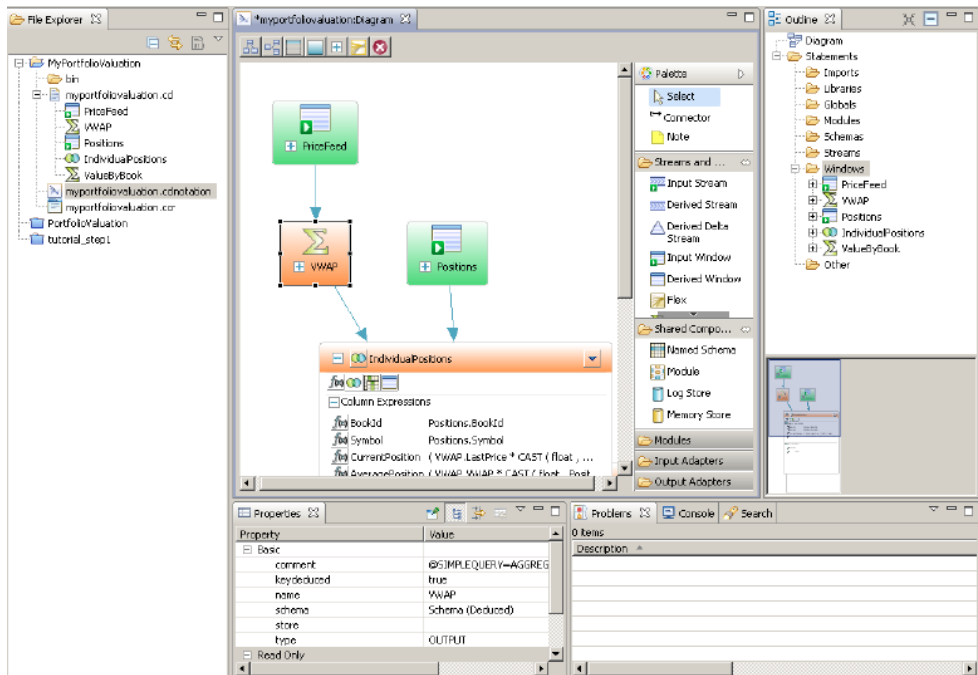
You can apply verbose and iconic mode to all elements in a diagram, or to the selected shape only.

Visual Authoring Environment

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

- **Editor** – canvas at the center of the Authoring perspective where you edit the diagram.
- **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.
- **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 validate a project or upload files.
- **Console** – displays messages generated by Studio scripts.

Figure 3: Authoring Perspective Views





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 the Sample Project* on page 20

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.

Connecting Elements

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

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


1. In the **Palette** view, select the **Connector** tool.

- Click the shape that will produce the output.

This attaches the connector line to the first shape.

- Click the shape that will receive the data to indicate the direction of data flow.

If a connection is allowed between shapes, you see a connection icon beside your cursor. If

a connection is not allowed, you see a "not allowed" icon .





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.

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.

Building a Simple Project

Build a simple project in the Visual editor.

The Portfolio Valuation example described here takes as input a set of positions (stock holdings) and a market price feed, and values the positions.

Sample Project Diagram

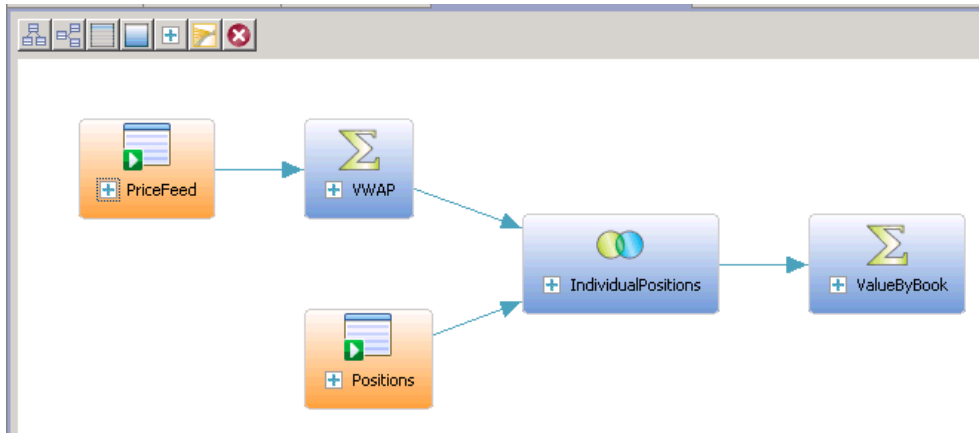
The Portfolio Valuation diagram uses simple queries to aggregate and join data from two input windows.

The example:

CHAPTER 3: Visual Editor Authoring

1. Reads data from an input window, PriceFeed, with five columns: Id, Symbol, Price, shares, and TradeTime.
2. Applies an Aggregate simple query to create a moving average—a volume weighted average price (VWAP). With the VWAP, you can see the value of positions based on the average price, rather than see the value of your positions change with every small price movement.
3. Read data from another input window, Positions, with three columns: BookId, Symbol, and SharesHeld.
4. Applies a Join simple query, joining the market price (from the VWAP aggregate) to your holdings (Positions), so that you can see the value of your position in each stock.
5. Applies one more aggregation to show the total value of each "book." This aggregate, ValueByBook, embodies a strategy where your positions may be organized by different "books," which may be portfolios or funds.

Figure 4: Portfolio Valuation Sample Diagram (Iconic Mode)



Creating the Sample Project

Use the Studio to define a new set of processing instructions for event data.

1. Select **File > New > Project...**
2. In the **Name** field, enter `my_portfolio_valuation`.

A valid project name:

- Must start with a letter, underscore, or dollar sign
- All other characters must be alphanumeric, underscore, or dollar sign
- Must not contain spaces

For your own projects you can use any name. To ensure that you can run the sample project you are creating, use the values listed here.

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.

Adding an Input Adapter


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

The input adapter identifies the external source for the input stream or window, and translates it into a format that Event Stream Processor Server accepts. You can add adapters to the diagram before or after adding input and output streams or windows.

This example shows you how to insert an adapter, enable it for schema discovery, then generate and attach the input window and its schema automatically. This is the best practice for adapters that support it.

Alternatively, ESP Studio allows you to create the stream or window and then attach an adapter. Use this method for adapters that do not support schema discovery, or where you want to explicitly create an inline schema for input streams or windows.

For a list of adapters that support schema discovery, and descriptions of properties to configure, see the *Studio Users Guide*.

1. Open the **Input Adapters** compartment in the Palette and locate the adapter you want. For this example, choose the **File XML Input** adapter, which reads data from an XML file.
2. Click the adapter in the Palette, then click in the diagram.
The adapter shape is inserted but its border is red, indicating it is not complete, until you define its properties and attach it to a stream or window.
3. In the adapter shape toolbar, click **Edit Properties** .
4. (Optional) In the Adapter Properties dialog, change **Name** to identify your adapter.
5. Under Adapter Properties, configure the adapter for schema discovery:
 - a) Choose **Set properties locally**.
This lets you set properties in the table on the right side of the dialog. Required properties are in red.

Note: Do not choose **Use named property set**, as it will not allow you to discover the schema for this adapter.

To use schema discovery for the File XML Input adapter, set the Directory and File properties to specify the absolute path to the data files you want the adapter to read.

- b) Click in the Value column for Directory, and click the button to browse for the directory with the data files. Then enter the File value.

For this example, specify the path to the sample data installed with the product.

Property	Value
Directory	<code>workspace_install_path\exampledata</code> Windows default: <code>C:\Documents and Settings\username\My Documents\SybaseESP\5.0\workspace\exampledata</code> Linux and Solaris default: <code>your_home_directory/SybaseESP/5.0/workspace/exampledata</code>
File	<code>positions.xml</code>

6. Press **OK** to save.

Next

Import the schema and create a connected input stream or window with the same schema as the data file.

See also

- *Input and Output Adapters* on page 7

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.

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

- *Discovering a Schema* on page 23


Discovering a Schema

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

Prerequisites

Add the adapter to the diagram and set its properties.

Task

1. Click **Schema Discovery**  on the adapter toolbar.
 - If the schema is configured properly and one or more data sets are found, 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.
2. Select the schema you need.


You can expand the data set to view the schema.

For this example, select **positions.xml**, then click **Next**.
3. In the Create Element dialog, choose **Create new input window**.

This option creates and attaches a new window to the adapter, creates an inline schema for the window, and populates the window with the schema discovered from the adapter.

When the adapter is not yet attached to a stream or window, other options are:

 - **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 new named schema.** – creates a new named schema and populates it with the schema discovered from the adapter.
4. Click **Finish**.

The new input window appears, and is automatically connected to the File XML Input adapter.
5. In the Schema compartment of the input window, click the **Toggle Key**  buttons for the BookId and Symbol columns to specify the primary key.

With the primary key, the shape becomes valid.
6. (Optional) Click the input window **Edit** button and name it `Positions`.

Next

Create another input window, PriceFeed. Either:

- Insert another File XML Input adapter, that takes input from the `pricefeed.xml` file in the same `exampledata` directory. Then use schema discovery to generate and connect the window automatically, and set the `Id` column as the primary key, or,
- Create the PriceFeed input window manually.

See also

- *Schema Discovery* on page 22

Adding an Input Window Manually


Add an input window to the diagram in the sample PortfolioValuation project.

These steps let you create an Input Window directly, and define the schema, without importing a schema.

If you used the input adapter to discover the schema and generate both input windows automatically, skip these steps and go directly to the next task.

1. In the Visual editor, in the Palette to the right of the diagram, open the **Streams and Windows** compartment.
2. Click **Input Window**.
3. Click in an empty area in the diagram where you want to insert the input window
The input window object is added to the project. The red border indicates that it needs more definition to be valid.
4. To set the name of the input window, either:
 - In iconic mode, click once to select the shape, then click again to edit the name.
 - In verbose mode, click the edit icon next to the name.

For this example, enter the name `PriceFeed`.

5. Expand the shape to verbose mode, and click **Add Column**  on the input window shape toolbar, to add each new column.


Tip: Hover over any icon to see its name.

A new column is created with a default name, and default datatype of integer.

6. Enter the column name, then click the datatype. Click again to choose a datatype from the drop-down.

For this example, enter these column names and datatypes:

- **Id** – integer
- **Symbol** – string
- **Price** – float
- **Shares** – integer
- **TradeTime** – date

7. Click the  button for the Id column to toggle it to the Key symbol.

A primary key is required for input windows.

The Id column is now the primary key for the PriceFeed input window. The red border should change to gray.

8. Save (**Ctrl+S**).

The input window and its schema (or deduced schema) are in the diagram. Because you did not set a retention policy, the default policy, save all records, applies.

Adding Simple Continuous Queries

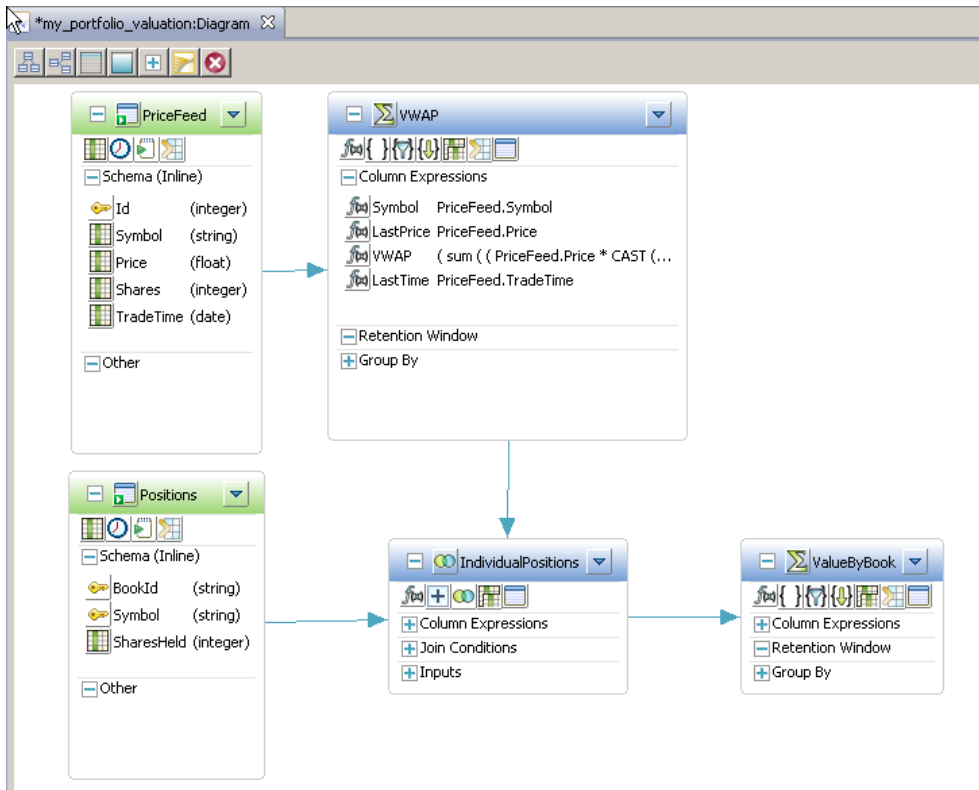
Review the types of queries available in the Visual editor. To get started, try adding one of the simple queries to your project.

To create the PortfolioValuation diagram, add three simple queries:

1. An Aggregate simple query called VWAP to create a volume weighted average price (VWAP). This aggregate should take input from the PriceFeed input window.
2. A Join simple query to join the VWAP output to the Positions input window. Name the join result IndividualPositions.
3. A second Aggregate simple query, ValueByBook, to show the total value of each "book."

For example details, refer to the figure, and the steps for creating and modifying simple queries.

Figure 5: Portfolio Valuation Sample Diagram (Verbose)




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.
- **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.
- **Join** – allows you to combine records from multiple streams or windows, forming a new record with information from each source.
- **Compute** – allows you to create a new event, with a different schema from the input, and compute the value to be contained in each column (field) of the new event.
- **Union** – allows you to combine multiple streams or windows that all share a common schema into a single stream or window.

-  **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.

Creating an Aggregate as a Simple Query

Add to the sample diagram an Aggregate simple query to create a volume weighted average price (VWAP).

An Aggregate query groups events that have common values, and computes summary statistics for the group.

1. In the Visual editor Palette, in **Streams and Windows**, click **Aggregate** ()
2. Click in the diagram to create the object.
3. Change the default name, Aggregate1, to VWAP.
4. Connect PriceFeed to the VWAP aggregate:
 - a) Click the **Connector** shape in the Palette.
 - b) Click the **PriceFeed** input window, then click the **VWAP** aggregate.
The aggregate border changes from red to black, indicating that it is valid, now that it has input.
5. Enter Column Expressions:
 - 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 Expressions** () in the shape toolbar.


For this example, copy these columns:

- PriceFeed.Symbol
- PriceFeed.Price
- PriceFeed.TradeTime

Then add another column and edit its name to VWAP.

6. Edit column expressions by double-clicking to open the inline editor, or by selecting the expressions and pressing **Ctrl+F2** to open the expression in the pop-up editor.
For this example, make these changes:
 - Edit Price to LastPrice
 - Edit TradeTime to LastTime
 - Edit the VWAP column expression to:

```
( sum ( ( PriceFeed.Price *
  CAST ( FLOAT , PriceFeed.Shares ) ) ) /
  CAST (FLOAT , sum ( PriceFeed.Shares ) ) )
```


7. Click **Add GroupBy Clause** () to edit the grouping of columns in the aggregate object.

Note: The Aggregate shape must have exactly one GROUP BY expression.

For this example, select **PriceFeed.Symbol** as the grouping column.

8. (Optional) Click **Set Keep Policy**  to create a retention window.

The default policy is to keep all rows of incoming data. You can also choose to keep only the last row, a specific number of rows, or keep the rows for a specific time. This defines the CCL **KEEP** clause.

9. (Optional) Use the Toggle  option to change the aggregate object from LOCAL to OUTPUT.

Creating a Join as a Simple Query

Add to the sample diagram a join simple query that combines multiple datasources, similar to the join expression options of the CCL **FROM** clause.

1. In the Visual editor Palette, in **Streams and Windows**, select **Join**.

If necessary, close the compartments below **Streams and Windows**, or use the arrow below the compartment, so that **Join** is visible.

2. Click in the diagram to create the object.


For this example, edit the join object name to be `IndividualPositions`.

3. Connect the join object to the appropriate stream or window.


Attach join objects to any stream, window, or Flex operator. Join objects have multiple inputs, but only one output.

Note: Streams, windows 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 more about supported joins, see the *Studio Users Guide* or the *CCL Programmers Guide*.

For this example, connect the VWAP aggregate object and the Positions input window to the `IndividualPositions` join object.

4. Click **Copy Columns**  in the join shape toolbar and select columns to copy.

For this example, choose **Select All**, then clear the check box on the second **Symbol** field.

5. Click **Add Column Expressions** .

For this example add two columns: Current Position and Average Position.

6. To modify Column Expressions, either:

- Press **F2** to open the inline editor. A drop-down menu displays which is populated with input columns and built-in functions. Select the input column or built-in function you want to use to define the expression.
- Press **Ctrl+F2** to open the expression editor. Press **Ctrl+Space** to display the available input columns and built-in functions, or enter the desired expression manually.
- Modify the expression in the Properties view.

For this example, create these Column Expressions:

- `BookId Positions.BookId`

- `Symbol Positions.Symbol`
 - `CurrentPosition (VWAP.LastPrice * CAST (float , Positions.SharesHeld))`
 - `AveragePosition (VWAP.VWAP * CAST (float , Positions.SharesHeld))`
7. In the Join Conditions compartment of the join shape, set up the join conditions. In this example, if you connected the join to the VWAP and Positions inputs, in that order, there are now two elements in the Join Conditions compartment. The first defines the leftmost element for the join. If you connected to VWAP first, the first element (left side of the join) is VWAP. You can double click it to change it. For this example, you must configure the second join element.
- a) Double-click the second join element to open the Edit Join Expression dialog.
 - b) Select the columns from Source 1 and Source 2 to join on.
For this example, select **Symbol** for both Positions and VWAP.
 - c) Click **Add**.
The columns chosen appear in Join Constraints, where you should now see `Positions.Symbol=VWAP.Symbol`. You cannot edit join constraints manually in the Visual editor.
 - d) (Optional) Choose a join type.
For this example, use the default, **LEFT**.
8. Click **OK**.

Completing the Sample Project

Clean up the diagram by removing unused elements.

1. Create an additional Aggregate Simple Query called ValueByBook,
 - a) Set column expressions:
 - `BookId IndividualPositions.BookId`
 - `CurrentPosition sum (IndividualPositions.CurrentPosition)`
 - `AveragePosition sum (IndividualPositions.AveragePosition)`
 - b) Set Group By to `IndividualPositions.BookId`.
 - c) Toggle to OUTPUT.
 - d) Connect it to the IndividualPositions join object.
2. Delete the unused input stream element NEWSTREAM that was added automatically when you created the project.
Remove any other unused elements as well, so that you can run the project.
3. (Optional) Toggle to Iconic mode to Compare your diagram to the figure in Portfolio Valuation Sample Diagram.

CHAPTER 3: Visual Editor Authoring

- Click the Toggle Image button in the upper left corner of a shape, or,
- Click the All Iconic or All Verbose button in the toolbar.

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.

CHAPTER 4 CCL 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.

The *Studio Users Guide* explains use of the CCL editor within ESP Studio.

For CCL language usage and reference 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.

By default, the `.ccl` file opens in the CCL editor, and the `.cclnotation` file opens in the Visual 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.

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.

CCL for the Sample Project

The CCL for the Portfolio Valuation sample project created in the Studio Visual editor is shown here, with line breaks added for readability.

```
CREATE INPUT WINDOW PriceFeed
  SCHEMA ( Id integer , Symbol string , Price float , Shares integer ,
  TradeTime date )
  PRIMARY KEY ( Id ) ;

/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW VWAP PRIMARY KEY DEDUCED AS
  SELECT
    PriceFeed.Symbol Symbol ,
    PriceFeed.Price LastPrice ,
    ( sum ( ( PriceFeed.Price * CAST ( FLOAT ,
PriceFeed.Shares ) ) ) ) /
    CAST ( float , sum ( PriceFeed.Shares ) ) ) VWAP ,
    PriceFeed.TradeTime LastTime
  FROM PriceFeed
  GROUP BY PriceFeed.Symbol ;

CREATE INPUT WINDOW Positions
  SCHEMA ( BookId string , Symbol string , SharesHeld integer )
  PRIMARY KEY ( BookId, Symbol ) ;

/**@SIMPLEQUERY=JOIN*/
CREATE OUTPUT WINDOW IndividualPositions
  SCHEMA ( BookId string , Symbol string , CurrentPosition float ,
AveragePosition float )
  PRIMARY KEY ( BookId, Symbol )
  AS
  SELECT
    Positions.BookId BookId ,
    Positions.Symbol Symbol ,
    ( VWAP.LastPrice * CAST ( float , Positions.SharesHeld ) )
CurrentPosition ,
    ( VWAP.VWAP * CAST ( float , Positions.SharesHeld ) )
AveragePosition
  FROM Positions , VWAP
  WHERE Positions.Symbol = VWAP.Symbol ;

/**@SIMPLEQUERY=AGGREGATE*/
CREATE OUTPUT WINDOW ValueByBook PRIMARY KEY DEDUCED AS
  SELECT
    IndividualPositions.BookId BookId ,
    sum ( IndividualPositions.CurrentPosition ) CurrentPosition ,
    sum ( IndividualPositions.AveragePosition ) AveragePosition
  FROM IndividualPositions GROUP BY IndividualPositions.BookId ;
```

ESP Studio lets you run and test all aspects of a project.

During development, you can use ESP Studio to run any compiled project against a local or remote server, view data flowing through the streams and windows defined in the project, execute queries, and use debugging tools. Your project configuration and licensing determine the type of server connections you can use when running projects. Some adapters also have special licensing requirements.

If your license supports it, in ESP Studio you can connect immediately to a local cluster to run projects.

In a production environment, you typically run projects on a remote server using the command-line utilities and procedures discussed in the *Administrators Guide*, rather than ESP Studio.

Run-Test Perspective

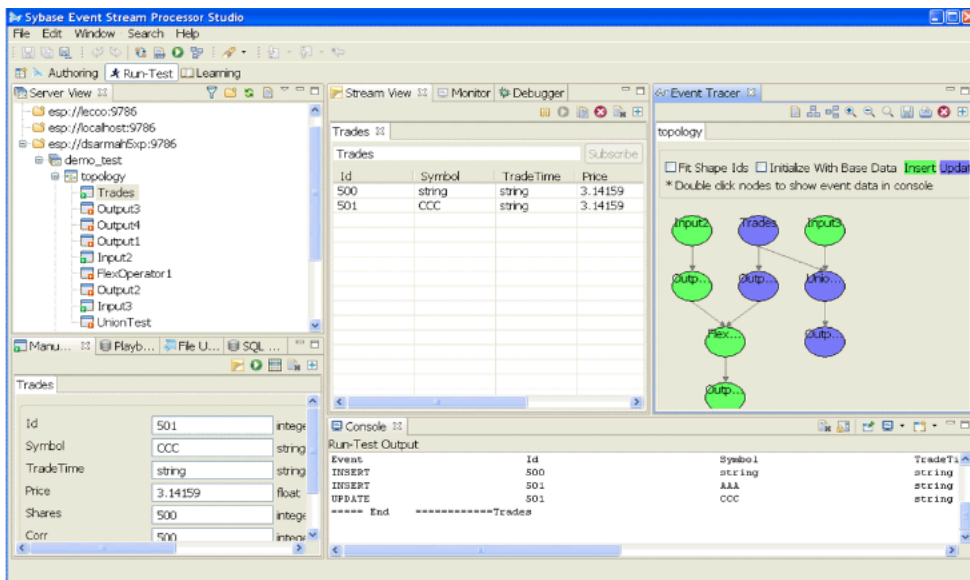
In the Run-Test perspective, access the tools used to test, debug, and fine-tune a project.

In Run-Test perspective you can:

- Start and connect to servers from the Server View
- Run projects
- Enter test data by uploading data files to a server in File Upload view, or manually entering data to a stream in Manual Input view
- Subscribe to a stream in Server View
- Publish data from a stream or window
- Execute a query against a running project in SQL Query view
- Use the Event Tracer and Debugger to set breakpoints and watchpoints, and trace the flow of data through a project
- Monitor performance in Monitor view
- Record in-flowing data to a playback file and play the captured data back into a running Event Stream Processor instance, in Playback view

The figure shows a sampling of view contents you might see in Run-Test perspective while testing a project. They do not represent the sample project created in this guide.

Figure 6: 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*.

Connecting to a Local Cluster

Connect ESP Studio to a local cluster.

Server connections appear in the **Run-Test** perspective.




1. Select the **Authoring** perspective.
2. Select **Run Project**  and select a project. Click **OK**.
The Server View in the Run-Test perspective opens, showing the project connection. A successful connection displays 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.

ESP Studio acts as a node (cluster manager) and automatically connects the project to the local cluster.

Viewing the Data in a Stream or Window

In the Stream View, watch data flow through a stream or window in a dynamic view that updates as the information changes.

1. In the Run-Test perspective, select the stream or window from the Server View.
2. Right-click on the output stream or window and choose **Show in > Stream Viewer**.
A tab opens in the Stream View displaying all new events. If you selected a window, all retained rows currently in the window are displayed.
3. (Optional) Select a subscription and choose an option, using the buttons at the top of the Stream View:
 - **Close Subscription URL**  — to disconnect and close the current Stream View.
 - **Clear contents and pause subscription** .
 - **Show current subscription in new view**  — to open a new Stream View with the current table visible.

Other Tools for Running and Testing Projects

ESP Studio includes many other tools for testing projects, including those in the Run-Test perspective.

For information beyond the scope of this guide on running, configuring, monitoring, querying, and debugging projects, see the *Studio Users Guide*.

Index

A

adapters

- attaching in Visual editor 21
- creating an input stream 23
- creating an input window 23
- custom 8
- discovering a schema 23
- importing a schema 23
- overview 7
- schema discovery 22

aggregate

- creating 27
- example 27
- simple query 26

APIs

- supported languages 8

attaching

- adapters 21

Authoring perspective

- views 16

C

CCL

- editing 31
- overview 9
- sample code 32

CCL editor

- overview 31

compute

- simple query 26

connecting

- adapters 21
- shapes 18
- starting a server connection 34
- to a local cluster 34

custom adapters

- overview 8

D

data flow

- viewing in Stream View 35

data-flow programming

- example 3

introduction 3

databases

- compared to Sybase Event Stream Processor 2

deleting

- elements from a diagram 30
- elements from a project 30

diagrams

- deleting elements 15, 30
- example 19
- iconic mode 19
- inserting shapes 18
- modifying layout 19
- overview 15
- verbose mode 19

discovering

- schemas 23

E

editing

- Visual editor 17

editing CCL

- CCL editor 31
- text editor 31

Event Stream Processor

- components 6

event streams

- overview 2

events

- delete 5
- examples 2
- insert 5
- update 5

examples

- overview 12

external data

- input and output adapters 7

F

filter

- simple query 26

filtering

- metadata streams 34

Index

G

GUI authoring
See visual authoring

I

iconic mode
 toggling 19
importing
 schemas 23
input adapters
 overview 7
 See also adapters
input windows
 adding manually 24

J

join
 simple query 26
joins
 creating 28
 example 28

L

layout
 modifying 19

M

metadata streams
 filtering 34

N

named schema 5

O

opcodes
 defined 5
 insert, update, and delete events 5
output adapters
 overview 7
 See also adapters

overview 9
 Sybase Event Stream Processor 1

P

pattern
 simple query 26
projects
 building simple projects 19
 creating 20
 deleting elements 30
 diagrams 15
 files 20
 introduction 3
 running 33
 running in local cluster 34
 simple example 20
 testing 33

Q

queries 25
 continuous 26
 simple 26
 See also simple queries

R

removing elements
 from a diagram 30
 from a project 30
Run-Test perspective
 overview 33
running a project
 in local cluster 34
running projects
 overview 33

S

sample CCL code
 Portfolio Valuation sample project 32
sample diagram 19
sample projects
 loading 13
samples
 See examples
schema
 adapters 22

- creating an input stream 23
 - creating an input window 23
 - discovering a schema 23
 - discovery 22
 - importing a schema 23
 - overview 5
- schema discovery
 - adapters 23
 - creating an input stream 23
 - creating an input window 23
 - importing a schema 23
 - overview 22
- SDKs
 - supported languages 8
- Server View
 - overview 34
 - showing servers in Event Tracer View 34
 - showing servers in Monitor View 34
- servers
 - connecting to 34
- shapes
 - iconic and verbose 19
 - inserting in a diagram 18
- simple queries 25, 27, 28
 - aggregate 27
 - descriptions 26
 - join 28
- SPLASH
 - overview 9
- Stream View
 - viewing data flow 35
- streams
 - introduction 4
 - schema 5
 - schema discovery 22
 - structure 5
- Studio
 - overview 8
 - starting 11
- Studio workspace
 - basics 11
- subscriptions
 - viewing data flow 35

T

- testing
 - projects 33

- text authoring
 - overview 8

U

- union
 - simple query 26

V

- verbose mode
 - toggleing 19
- viewing data flow
 - in Stream View 35
- views
 - Authoring perspective 16
 - Run-Test perspective 33
 - Stream View 35
- visual authoring
 - diagrams 15
 - overview 8
 - views 16
- Visual editor
 - accessing 17
 - aggregate simple query 27
 - creating dataflow 18
 - join simple query 28
 - modifying layout 19
 - overview 15
 - simple queries 26
 - views 16
- VWAP
 - example 19
 - simple query example 27

W

- windows
 - adding manually 24
 - introduction 4
 - schema 5
 - schema discovery 22
 - structure 5
- workspace
 - basics 11

