Continuous Intelligence™ with Complex Event Processing

When you need to understand and respond, as fast as things happen
This technical white paper introduces the Sybase Event Stream Processor (ESP), a platform that use Complex Event Processing (CEP) to deliver real-time analysis of fast moving data. It covers the applications of CEP for real-time insight, and then provides a comprehensive overview of the ESP technology and architecture.

<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Introduction</td>
</tr>
<tr>
<td>1 Section 1: Introduction to CEP</td>
</tr>
<tr>
<td>1    Volume, Velocity, Variety: the Need for Complex Event Processing</td>
</tr>
<tr>
<td>1    Complex Event Processing (CEP)</td>
</tr>
<tr>
<td>4    Examples of Applications Built with Sybase® ESP</td>
</tr>
<tr>
<td>5  Section 2: The Sybase Event Stream Processor</td>
</tr>
<tr>
<td>5    Key Concepts</td>
</tr>
<tr>
<td>6    Architecture Overview</td>
</tr>
<tr>
<td>9    Integration</td>
</tr>
<tr>
<td>10   The ESP Studio – Maximizing Productivity</td>
</tr>
<tr>
<td>13   CCL – The Continuous Computation Language</td>
</tr>
<tr>
<td>20   Part of a Comprehensive Real-time Analytics Platform</td>
</tr>
<tr>
<td>21   Sybase ESP+RAP in the Capital Markets</td>
</tr>
</tbody>
</table>
INTRODUCTION

The Sybase Event Stream Processor (ESP) is a high-performance complex event processing platform designed to make it easy for firms to implement continuous intelligence solutions. Sybase ESP 5.0 is the latest version of our CEP product line, building on the success of its predecessors, the Aleri Streaming Platform and the Coral8 engine (later known as Sybase CEP). It is a comprehensive platform for rapid application development and deployment, with a scalable high-performance CEP engine that applies continuous queries to streams of incoming events in real-time, producing insight from raw data continuously and immediately. This white paper is divided into two main sections: the first is an introduction to Complex Event Processing and how it is used; the second focuses on the Sybase Event Stream Processor and its architecture and features.

SECTION 1: INTRODUCTION TO CEP

Volume, Velocity, Variety: the Need for Complex Event Processing

Businesses today both benefit from and are burdened by the wealth of information at their disposal. Systems that automate and facilitate every aspect of the business operation produce, collect and send information — and at ever increasing volumes and rates. In the capital markets, market price data is the life-blood of trading and it arrives at the rate of 100’s of thousands of messages per second. Interacting with customers via a website produces a steady stream of “clicks” as the user interacts with the site. RFID sensors produce streams of messages tracking the movement of goods.

Making sense of all this data can be daunting. There’s the inherent complexity of combining a variety of data from a diverse set of sources, and then the analysis to extract actionable insight from the raw data. The traditional approach to this is to collect all the data in a database or data warehouse and then use business intelligence (BI) tools to analyze the data.

Increasingly, however, there is business value in being able to respond faster to new information. Velocity has value. Rather than analyzing historical data to understand what happened last week or even yesterday, what if you could understand what is happening right now? Even better, what if your systems could tell you when something is happening that deserves your attention? What if your systems could automatically respond to the new information, leaving you out of the loop entirely?

Application that continuously absorb, analyze and respond to streams of information as fast as things happen are delivering Continuous Intelligence, enabling “right now” response across a range of industries.

Complex Event Processing (CEP)

Consider the following situations...

• An automated trading application that scans massive amounts of incoming market data to spot trading opportunities, where the trigger to trade has to be instantaneous or the opportunity is missed.
• A market making application that has to adjust internal or published rates in response to market movements — delays either mean lost business or lost profit.
• A risk management application that continuously updates aggregate position and risk information, combining data from multiple systems to provide a single consolidated view that is always current.

These are just a few examples of the types of applications that can benefit from complex event processing technology. The common denominator among these applications is that they share the need to continuously collect, process, and analyze data in real-time, producing results without delay, even when the data arrives at very high rates.

While traditional databases were designed to process individual transactions at very high rates; analyzing the data to look for specific conditions or deriving higher level summary data were tasks that had to happen “off-line,” using query tools that were never designed to produce actionable intelligence in real-time. This made them unsuitable for applications that had to analyze data in real-time (such as trading applications), and while they were adopted for data analysis in other areas, the fact that the analysis is done on “historic” data means that the business insight is derived after-the-fact, in many cases missing an opportunity to react quickly to the results of the analysis.
Complex event processing technology delivers the data analysis tools traditionally provided by relational databases or even spread sheets, but in a real-time event-driven implementation that is capable of processing incoming data at very high rates and producing results with near-zero latency.

Think of it as taking some of the fundamental concepts of a relational database and turning them upside down: a traditional relational database is designed to collect data and store it, where you can then analyze it to filter the data, combine it, group it, search for patterns, derive high level summary data, etc. The analysis happens off-line, not in response to incoming events. An event processor (the heart of CEP technology), in contrast, takes incoming messages and runs them through a set of pre-defined continuous queries to produce derived streams or sets of data.

We call them continuous queries because the data analysis logic is similar to what might be included in a traditional database query. For example:

- Show me which events meet this criteria
- Tell me if this pattern of events (or non-events) occurs
- Show me the current total of all events matching this criteria
- Group events by these values and calculate the average for each group

While complex event processing logic may be similar to a traditional database query, the implementation is anything but. Event processing uses a dataflow architecture to pass incoming messages through the continuous query operators as soon as the message arrives so that the result sets are instantly updated. These functions used within the continuous queries have been implemented in a way to maximize throughput and minimize latency.

An Alternative to Custom Code

Complex event processing technology provides an alternative approach to building high performance enterprise-class applications that have to process event data in real-time. Custom applications written in C++ or Java are expensive, time consuming to build, and are typically inflexible and therefore expensive to maintain since the processing logic is hard coded and tightly bound to the data structures. What’s more, designing and writing highly efficient code for real-time processing requires specialized programming skills.

In fact our users have told us that, on average, they can build and deploy new applications using Sybase CEP technology in 20% of the time that it would have taken to build the applications in Java. That’s an 80% reduction in cost and it means getting the new application into products in one fifth of the time. And that’s only the initial benefit. Once the application is in production, there is far less code to maintain, meaning less ongoing support costs.

The Need for Agility

A by-product of building applications on a CEP platform is greater agility. Because the business logic is implemented using high level authoring tools, and is separated from the underlying data handling infrastructure, the business logic can be rapidly changed as the needs of the business change. Customers using applications built on CEP can literally change the processing logic in minutes. And because CEP is based on an event driven architecture, a CEP-based application is decoupled from the inputs and outputs, reducing dependencies across the various systems that produce and consume data.

A Spectrum of Uses and Requirements

Regardless of the specific terms used, all event processing applications set out to do one or more of the following:

- **Situation Detection:** Monitor incoming events to detect patterns that indicate the existence of an opportunity or a problem — i.e. a situation that warrants a response or that needs to be recorded. This can range from a simple filter to a complex set of rules that correlate incoming events and screen for sets of conditions that may include missing events. High level events indicating the existence of the situation are generated as the result.
• **Data Aggregation and Analysis — Continuous Computation:** Data is correlated, grouped and aggregated, and computations are then applied to produce new information such as summary data, high level statistics, or adjustments to key operating parameters. Examples of this type of CEP include:
  - Continuously adjusting prices based on movements in the market or other real time inputs
  - Continuously updated key performance indicators (KPIs)
  - Continuously update valuations, exposures
  - Continuous aggregation of data from multiple sources to see "the big picture"

• **Data Collection:** A by-product of CEP is often the collection of raw event data and/or higher level summary data. The collected data can be used as context for processing newly arriving events and can also be stored in an historical database for off-line analysis, reporting or to have an audit trail.

• **Application Integration, Intelligent Event Handling:** Many applications are built on an Event Driven Architecture, but the basic tools for EDA provide the mechanisms for the exchange of event data without providing the ability to analyze event data. CEP can provide intelligence within an event driven architecture to analyze events in the context of other events and a knowledge of the state of various systems to determine what new events need to be generated or to determine the action to be taken based on an event.

This is an important point, since in a particular context you may find the focus to be on one particular aspect of event processing. Yet recognizing that different applications have different needs will help you ensure that you select the optimal tool or tools for the job.

---

**Design Patterns for Complex Event Processing**

When we talk about the ability to analyze incoming event data in real-time, we are actually referring to a variety of functions that can be applied to the data, alone or in combination, to derive high level intelligence and/or to trigger a response. Examples include:

• Filter data to apply simple or complex filters to detect conditions of interest. This can include correlation of events across multiple sources, correlation of events across time, and watching for sets of events that match a defined pattern.

• Combine data from multiple sources, including the ability to combine streaming and static data or to combine data that arrives at different times. Define data retention "windows", either based on time or number of elements, across which the computations will be performed.

• Group and aggregate data, producing high level summary data and statistics. This can include trends (moving averages), net positions/exposures, etc.

• Compute new data elements: enrich simple event data by adding new fields that are computed based on context, data from other sources, etc.

• Watch for specific patterns of incoming events, generating a new event when the pattern is detected. The new event can be used to trigger a response or to generate an alert.

• Transform data format and structure. This can go beyond simple message-level transformation and can create entirely new events based on individual or multiple events using rules that take into account context, reference data, etc.

• Generate high level events from patterns or groupings of low level events.

---

**Not All Real-Time is the Same: Event-Driven vs. On-Demand**

When we talk about event processing, we are generally focused on the real-time aspect of it. While technically, event processing can be applied to historical data, in general the focus of event processing, and certainly the Sybase focus on event processing, is analyzing streams of event data in real-time. After all, if you’re not looking for insight and response as fast as things happen, Sybase already offers powerful technology for collecting, managing and analyzing historical data. It’s the real-time aspect that sets event processing apart from the other options.
But there are many products and technologies that offer real-time capabilities. After all, even Sybase IQ offers a real-time loading option, and Sybase RAP is a real-time analytics platform. The key distinction therefore is not just that Sybase ESP analyzes data in real-time, but that it is event driven. It's the difference between on-demand and continuous. Analytics platforms such as Sybase IQ and Sybase RAP collect, store and manage data so that it can be analyzed when you want to analyze it. Sybase ESP analyzes the data as it arrives, and either initiates a response or notifies you that something has happened — or, even better, is happening or is likely to happen soon.

Sybase ESP is designed for speed and scalability, so that it can process and analyze data as fast as it arrives — even if it’s coming in at the rate of hundreds of thousands of events per second, and deliver the results within milliseconds of the arrival of the new information. The bottom line: not all real-time is the same.

Examples of Applications Built with Sybase ESP

Sybase has customers using CEP technology in a wide range of different applications across a number of industries. Here's just a sample of some of the CEP-based applications built and by Sybase customers:

**Capital Markets:**

*Pre-Trade*
- Market analysis: Combine, analyze and enrich market data, covering trades, quotes and depth, to provide real-time insight and feed automated trading strategies with actionable data
- Automated security pricing: Continuous quoting
- Order validation and pre-trade limit approvals

*Trade*
- Automated trading strategies: Both alpha strategies and automated execution
- Trade monitoring: Real-time monitoring of trading algorithms and/or traders
- Smart order routing: Incorporating dynamic routing rules to adapt to the market and ensure best execution
- Real-time Surveillance and compliance monitoring

*Post-Trade*
- Real-time P&L: Using live market prices applied to continuously updated net positions
- Real-time Risk Monitoring: Continuous consolidation and aggregation of positions, real-time valuation of positions and exposures, limit management

**Financial Services:**
- Bank liquidity management: Continuous monitoring of actual and expected cash flows in bank treasuries
- Fraud detection/prevention: Real-time detection of event patterns indicating possible fraud

**Telecommunications:**
- Real-time system/network monitoring: Traffic volumes, problem detection
- Fraud detection: Spotting unusual traffic patterns in real-time
- Customer retention: Proactive customer service

**eCommerce:**
- Click stream analysis for real-time offer optimization, fraud detection, trend analysis, and customer experience management

**Energy:**
- Smart meter monitoring
- Demand management and response

**Public Security:**
- Bridge monitoring using remote vibration sensors to detect structural issues

**Healthcare:**
- Emergency room patient flow monitoring
- Procedure compliance monitoring and alerting
THE SYBASE EVENT STREAM PROCESSOR

The Sybase Event Stream Processor (ESP) is a high performance enterprise-class complex event processing engine that can be used to quickly implement and deploy a wide range of applications that need to analyze and act on event data in real-time. It is fifth generation CEP technology and the successor to the successful Aleri Streaming Platform. It represents the state-of-the-art in complex event processing, combining performance, versatility, and ease-of-use in an enterprise-class implementation designed for use in the most demanding environments.

Key Concepts

The Sybase Event Stream Processor is designed to process and analyze streams of events. So a natural starting point is:

An Event: in the real-world an event is something that happens. In the ESP context, and event is actually a message that contains information about a real-world event. An event could be an order arriving in a database, a price update on a stock market ticker, a user clicking on a web link, or a sensor reporting a piece of information.

Each event (message) has a schema: a set of fields, with a name and datatype associated with each field. In database terminology, an event is a tuple. In fact, because ESP adopts many concepts from databases, we refer to the fields in an event message as columns.

An event stream can be thought of as a channel that delivers events that have a common structure or schema. In ESP terminology, we generally refer to these simply as streams.

Thus streams of event data are a key starting point for any ESP project. Event streams flow into the ESP server into either an Input Stream or an Input Window.

Streams in an ESP project are stateless: a single event enters, passes through the stream, and is “forgotten” once it is passed downstream.

Windows in an ESP project have state. They are very similar to a table in a database, except that they are constantly changing. Incoming events either add a row to the window, update an existing row in the window, or delete a row from the window. The columns in the window match the schema of the incoming events. The size of the window can be set in three ways:

1. Number of rows: a window set to a fixed number of rows will shed older rows when new rows are added once the size limit is reached.
2. Time: a window with a time policy will keep rows for the defined time period and then delete them. So a one minute window will keep each row for one minute after it arrives (or is last updated) and then delete it.
3. Self-managed: since events don’t have to just add rows to a window, but can also modify or delete existing rows, a window can be self managed. No size or time limit is set, but rather incoming events will delete existing records in the window. An example of this would be an order book which has a set of open orders. New orders get added to the book and when an order is filled or cancelled an incoming “cancel” or “fill” event removes it from the book.

All Windows have Primary Keys. A primary key consists of one or more fields in the row that uniquely identify that row in the table.

Events can (optionally) have explicit operators — or OpCodes. This is a unique aspect of the Sybase Event Stream Processor, though not at all unique in the broader scope of data processing. Incoming events may directly update information maintained in a Window, and they can contain an opcode that indicates how they should be applied to the window. Thus an insert event adds a row to the window, an update event updates an existing row with the same key, and a delete event deletes the row with the indicated key value.
Continuous Queries take input from one or more Streams or Windows and apply an operation to produce a new stream or window. Stream and Windows that are produced as the result of a continuous query can be output streams/window or they can be local streams/windows, performing an intermediate step on the overall processing of the event.

An ESP project is what runs on an ESP Server. It defines a set of input streams/windows and a set of continuous queries that produce one or more output streams/windows.

An ESP application is a complete set of components that addresses a business need. It will consist of one or more projects, one or more adapters, and will often include other components such as a user interface, a database, etc.

Architecture Overview
Sybase Event Stream Processor is a platform for building applications that need to process and analyze streams of event data in real-time. It consists of a number of components:

- The ESP Server (and Cluster Manager): ESP projects run on an ESP Cluster. The cluster is managed by a cluster manager that receives and manages requests to start and stop projects. Each project runs in its own container — an independent server process that is started by the cluster manager. The cluster manager is also responsible for re-starting projects after a failure of the project or the machine it’s running on. Any number of cluster managers can run — all are peers, which avoids a single point of failure.
- The ESP Studio: The Eclipse®-based Studio is the integrated development environment to build and test ESP projects. It does not need to be running in production systems: once a project is built and tested with the Studio, while the project can be run from the Studio, it can also be deployed directly. For that matter, the Studio is entirely optional: project can be written in CCL using a text editor, and then compiled and deployed using command-line utilities.
- The ESP Compiler: Compiles CCL into ccx which can be run directly on the ESP byte-code engine. The compiler can be invoked from the Studio or the command line.
- The ESP Pub/Sub Interface: This is the interface that is used to stream data into an ESP project and/or subscribe to data being published by an ESP project. It can also be used to load static data into an ESP project. It is part of the ESP Software Developer’s Kit (SDK) which is available for C/C++, Java and .NET.
• **ESP Adapters:** Adapter are built on the pub/sub API and connect the ESP server to another application. While the ESP SDK can be used to build custom adapters, ESP comes with a wide range of built-in adapters for connecting to common environments. Sybase also makes a number of optional specialty adapters available as well. See the Appendix for a list of available adapters.

• **The ESP Ad-Hoc Query Interface:** Most ESP projects maintain windows of data (see “Concepts” section on page 7). While the primary purpose of ESP is to produce event-driven streams of information, there are many instances where it's useful to be able to run ad-hoc SQL queries against the windows to get a snapshot of the current set of data. ESP provides an ODBC/JDBC interface that allows windows to be queried by external applications.

---

**The ESP Cluster — A “Private Cloud” Architecture**

ESP projects run on an ESP Cluster. You can think of an ESP Cluster as an ESP “cloud” that can encompass any number of physical servers. When you run an ESP project, you run it on the ESP Cluster, rather than on a specific machine, and the ESP Cluster Manager assigns the project to a physical machine.

In its simplest form, an ESP “cluster” can consist of a single ESP Cluster Manager running on a single machine. When you instruct the cluster manager to run a new project, it will be started on the same machine. But as additional machines are added to the cluster, the cluster manager will make use of all available machines when starting new ESP projects.

Each ESP project runs as an individual process on an ESP cluster. When starting the project, the cluster manager starts an instance of the ESP server, running the desired project.

In a multi-machine ESP cluster, there will normally be two or more ESP Cluster Managers, running on different machines, to avoid a single point of failure. Cluster managers run as peers, so the loss of any one or more cluster managers does not affect the operation of the cluster as long as there is at least one cluster manager running.

Machines in the cluster that do not have a manager will have an “ESP Controller” that acts as an agent to manage local projects under the auspices of the cluster managers.

Some features of the ESP cluster include:
- A single cluster can be limited to a single machine or can span any number of machines
- An ESP cluster can run any number of projects simultaneously, subject to the capacity of the available hardware resources
- When a project is started, it is started on the ESP Cluster, not on a specific machine. The cluster manager assigns it to a machine.
- Connections to streams/windows in projects are made via a URL consisting of the server.workspace.project.element. The cluster manager resolves the URL to specific host and port.
- Projects run in workspaces, providing a namespace protection.
- Projects can be started/stopped independently of all other projects running on the server.
- Projects can be designated for automatic failover or active-active configurations (see below on Resiliency)
- The cluster manager is fully redundant — multiple cluster managers will run as peers, avoiding any single point of failure.
- Clusters can span multiple locations — the machines in a cluster do not have to be co-located. Intra-cluster communication is all TCP-based.
**Resiliency**

At the level of the ESP Cluster, as long as there is more than one machine in a cluster, there is no single point of failure. While a cluster can run as a single node on a single machine, where resiliency is desired there should be at least two machines in the ESP cluster, each with an ESP Cluster Manager.

Cluster managers run as peers — there is no primary or secondary — and there can be any number of cluster managers. The loss of one or more cluster managers will not affect the operation of the cluster as long as at least one cluster manager is still running.

**ESP Project Resiliency**

There are three options for resiliency when running an ESP project — the desired option is specified in the project configuration file (or can be specified at run-time):

1. **No Resiliency**: If the project fails it must be manually restarted.
2. **Automatic Failover**: If the project fails, the Cluster Manager will restart it. Project send heart-beats to the cluster managers to allow detection of a “hung” project.
3. **Active-Active**: Projects configured for active-active will run with redundant live instances, one as a primary and one as a secondary. Normally the primary and secondary will run on separate machines. All connections from outside the cluster are directed at the primary while it is alive. Data between the primary and secondary is continuously synchronized. If the primary fails, all outside connections will automatically be redirected to the secondary.

**Security**

Sybase ESP has built-in security features for authentication, access control, and encryption. User authentication is done at the cluster level, and there are three types of authentication that are supported:

- LDAP
- RSA
- Kerberos

ESP supports SSL encryption of all communication between the ESP cluster and client applications (including both data sources and consumers).
Role-based access control can restrict what individual users can do. Authorization can be used to restrict activities both at the cluster level and at the project level and, within a project, restrict access to individual streams and windows. Thus, access control can restrict, based on the authenticated identity of the user, whether they can:

• Administer the cluster
• Start/stop projects
• Write to or subscribe to a particular stream or window in a named project

Integration

One of the powerful aspects of Sybase ESP is the decoupled-nature of an event driven architecture. Integration of ESP into an existing environment is typically non-intrusive. ESP can go in as an overlay to existing systems without needing to replace or re-engineer those systems.

Integration with Sybase ESP is via input adapters and output adapters. ESP comes with a set of common adapters that are part of the base product. Sybase offers additional specialized adapters as add-ons. And custom adapters can be easily built using the Sybase ESP Software Developers Kit (SDK).

Adapters can run in-process as part of an ESP project, or they can run externally as a stand-alone process, in which case the communication between the adapter and the ESP server is via a TCP socket. External adapters can run on the same machine as the ESP project or on a different machine.

Standard Adapters

The Sybase ESP product ships with the most common adapters included. These simply need to be configured to connect to the relevant source or destination. We are constantly adding new adapters, so see the online list for adapters included in the product. These adapters include:

• Message buses: Connect to a message bus to receive incoming events and/or publish events. Supported message buses include JMS, TIBCO®, IBM Websphere MQ®
• Databases: Load data from and/or publish data to databases via ODBC/JDBC. Most major databases are supported. Input data can be loaded at startup or the database can be periodically polled.
• File: Read input events from a file; write output events to a file. With csv and xml parsing. One time or polled.
• Socket: Receive events on a socket; write events to a socket. With csv and xml parsing.

Specialized Adapters

A number of specialized adapters are available from Sybase as add-ons and are licensed separately. These include:

• Sybase Replication Server®: Real-time change capture for most commercial databases
• Market data: Including Thomson Reuters® and others
• FIX: (Financial Information eXchange)

Custom Adapters

Custom adapter can easily be developed using the ESP Software Developers Kit (SDK). Custom adapters can written in C, C++, Java, or .NET and can be built as either internal (C/C++ or Java only) or external adapters.

Custom adapter can also be developed without using an ESP API by simply writing events to and/or receiving events from a socket or file in an ESP supported format.
**Software Developers Kit (SDK)**

The Sybase ESP Software Developers Kit (SDK) is available for Java, C/C++, and .NET. It provides both publish and subscribe capabilities to allow custom adapters to be built. What’s more, while we generally talk about "adapters", in fact the ESP SDK can be embedded directly in an application that publishes to ESP, subscribes to ESP, or both.

The ESP SDK provides a simple way of:
- Connecting to an esp cluster
- Constructing an event with a schema corresponding to the stream/window that will receive it
- Publishing an event to a specific stream or window in a specific project
- Subscribing to an output stream or window in a live project
- Parsing the event received from an output stream/window

**The ESP Studio — Maximizing Productivity**

The ESP Studio is built on the Eclipse® framework, providing a familiar environment to many users. It provides capabilities to build, test, tune, debug and run ESP projects. It is designed around two key goals:

1. Easy to learn, even for non-programmers. While certain advanced tasks, such as writing custom operators in SPLASH, require programming skills, entire projects can be created and run by technically proficient users that aren’t programmers. The Studio is designed to be accessible and easy to learn by these users as well as experienced programmers.

2. Efficient to use. After all, ESP is a productivity tool. Therefore it’s important that the Studio is designed with user productivity in mind. The goal is for experienced users to feel that it’s an efficient environment in which to work.

**Two Editors: the Visual Editor and the CCL Editor**

The Sybase ESP Studio takes an innovative and unique approach to authoring by providing a choice of two different styles of editing in a seamlessly integrated environment. The Visual Editor allows the user to build a project by selecting building blocks from a palette, arranging them on a diagram, adding connectors to direct dataflow between, and configuring the behavior of each block. The CCL editor is a more traditional programming environment: a syntax-aware text editor. The unique aspect of the ESP Studio, however, is that the user can flip between the two editors at any time; changes made in one are reflected in the other.
Many users will choose to build their projects primarily in the visual editor, but may find it useful to occasionally look at or even directly edit the underlying CCL. Likewise, some users will find it more efficient to build projects directly in CCL, but will occasionally find it useful to switch to the visual editor to see the dataflows within the project. This is especially true for large/complex projects where the visual editor can serve as a useful navigation tool within the project.
Connecting to Data Sources

Most Projects begin with one or more data sources. Normally, at least one data source will be a true “stream” of events as they happen. There may be more than one streaming source, and there may be other sources that provide either static or semi-static data (i.e. data that updates infrequently).

The ESP server contains a number of built-in adapters that can be used to connect to common data sources. Sybase also provides a number of optional add-on adapters, and custom adapters can be built using the ESP Software Developers Kit (SDK). See the chapter later in this paper on adapters for more information.

Within the Studio, a user can select an adapter from the Palette, add an adapter to the Project, and configure the adapter. The adapter will then be controlled by the ESP server.

To simplify the process of building new projects that will use data from existing sources, the Studio includes a feature called Schema Discovery that lets the user easily select a schema from an existing source, import it into the Project and, if desired, create an Input Stream/Window connected to the source and with the schema for that source. Some adapters also provide features to allow for custom field mappings between the source and the input stream.

The Run/Test Perspective

The ESP Studio is a complete Interactive Development Environment (IDE) with tools for both authoring and testing. The Run/Test perspective includes tools to start/stop projects, monitor/view live projects, and test, debug and tune projects.

Specific tools include:
- Server View: Connect to any number of ESP servers. See running projects. Start/stop projects.
- StreamViewer: See the output from any stream/window, updated in realtime, in a tabular view
- Performance Monitor: Provides a graphical view of the project with color coding to indicate throughput rates or queuing. Updates dynamically, providing ability to identify bottlenecks.
- Event Tracer: Lets the user view how an event affects each query node in a project
- Debugger: Ability to set breakpoints in a project
- Record/Playback: Record data coming into input streams of a project, capturing it in a file that can then be played back any number of times. Playback speed can be the rate the data was originally received, sped up, slowed down, as fast as possible, or at a fixed rate.
- File Upload: Allows data to be loaded into input streams directly from data files
- Manual Input: User can manually input individual events, setting the value in each field
- Snapshot Query: Run a snapshot SQL query against a window in a live project and see the results
CCL — The Continuous Computation Language

Overview

The Sybase Continuous Computation Language (CCL) is the primary event processing language of the Sybase Event Stream Processor ESP projects are defined in CCL, which defines the inputs and a set of continuous queries to produce the desired outputs.

CCL is an adaptation of SQL with extensions for event streams. It provides the same type of sophisticated data analysis capabilities you find in SQL, including the ability to filter, group, aggregate and join data across streams. However, CCL also includes features that are required to manipulate data during real-time continuous processing, such as defining windows on data streams, watching for patterns of events, and invoking custom event handlers.

What really sets CCL apart from SQL is that it is designed for expressing continuous queries. A normal SQL query against a relational database executes once each time it is submitted to a database server and must be resubmitted every time a user or an application needs to re-execute 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 data sources.

Although CCL borrows SQL syntax to define continuous queries, the ESP server does not use an SQL query engine. Instead, ESP 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 Example

The following simple example shows how CCL is used to define two continuous queries:

```
CREATE INPUT WINDOW PriceFeed KEEP 10 min
SCHEMA (id integer, Symbol string, Price money, Shares integer, TradeTime date)
PRIMARY KEY (id);

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

CREATE LOCAL WINDOW VWAP
PRIMARY KEY DEDUCED
AS SELECT
  PriceFeed.Symbol Symbol,
  PriceFeed.Price LastPrice,
  (sum(PriceFeed.Price * PriceFeedShares)/sum(PriceFeedShares)) VWAP,
  PriceFeed.TradeTime LastTime
FROM PriceFeed
GROUP BY PriceFeed.Symbol;

CREATE OUTPUT WINDOW IndividualPositions
PRIMARY KEY (BookID, Symbol)
AS SELECT
  Positions.BookID BookID,
  Positions.Symbol Symbol,
  (VWAP.LastPrice * Positions.SharesHeld) CurrentPosition,
  (VWAP.VWAP * Positions.SharesHeld) AveragePosition
FROM Positions, VWAP
WHERE Positions.Symbol = VWAP.Symbol;
```
Common CCL Query Types

A CCL Project consists of one or more input streams/windows and one or more derived streams/windows. Each derived stream/window includes one or more continuous queries that take data from inputs and produce the derived result. Most continuous queries do some combination of the following operations:

- **Filter**: Filter an input stream, only passing through the events that pass the filter criteria
- **Aggregate**: Group incoming events according to a common “key” value, producing a single output record for each group. One or more fields in the output record will typically be computed using aggregate functions such as sum(), count(), average(), or even more advanced statistical functions such as standard deviation or correlation.
- **Compute**: Transform an incoming event into an event with a new set of fields where the new fields are computing from the fields of the incoming event, possibly along with other information
- **Join**: Join events/records from two or more inputs based on common field values. You can join a stream to a window, join two windows, and even do multi-way joins. This can be used to enrich incoming events with additional reference information, or it can combine data from multiple event streams, matching the events within a defined time window.
- **Pattern**: Watch for a specific pattern of events, and generate a new event when the pattern is detected. Patterns are defined with a time interval and a sequence or combination of events, and can include missing events — for example a pattern might be “eventA followed by eventB, but NOT eventC — within 10 minutes”

The CCL Query Graph – Dataflow Programming

An ESP Project can consist of a single input stream followed by a single continuous query, but a Project will often contain multiple continuous queries. Large Projects may contain dozens of inputs and hundreds of continuous queries.

ESP uses dataflow programming to direct incoming data into and through a set of continuous queries. Thus a project is broken down into a sequence of continuous queries, each of which further refines the incoming data into the set of desired outputs. Visually, such as in the ESP visual editor, it’s very easy to represent this as a set of streams and windows, with the data flowing between them. Thus a typical query graph of an ESP Project might look like this:

![Query Graph Example](image)

SPLASH Scripting Provides Extensibility

SQL is a great starting point for the most common functions used in processing event streams. It has the benefits of familiarity and the simplicity of a declarative language. But there are times when the event processing logic you need to implement can’t be easily expressed in a SQL query. That’s where SPLASH comes in. It brings extensibility to CCL.

SPLASH (Stream Processing Language Shell) is a simple procedural scripting language that has a syntax that is similar to Java while its spirit is closer to little languages like AWK or Perl that solve relatively small programming problems. It provides procedural control, the ability to iterate using WHILE and FOR loops, and provides data structures for holding data from one event to the next.
Unlike having to go outside of ESP to use Java or C++ to write custom functions or operations, SPLASH has the benefits of being:

- **Easy to learn:** As a small language, there is a small learning curve. You don’t need to be a Java programmer to use SPLASH.
- **Maximizes user productivity:** SPLASH is embedded inline in CCL files. No need to leave the ESP studio or use separate testing tools.
- **Efficient:** SPLASH is designed for high performance in processing incoming events, leaving the user to focus on the business logic.
- **Safe:** SPLASH only exposes the tools needed to process events, thereby limiting the opportunities for error.

### SPLASH Controls and Data Structures

SPLASH includes controls such as FOR, WHILE, and IF, giving the user complete procedural control with the ability to iterate over data sets. SPLASH includes the ability to define variables that retain state from one event to the next. It also includes advanced data structures that are designed for efficient handling of events and event sets including:

- **EventCache:** This is an alternate windowing mechanism for working with sets of events that allows greater control than the standard CCL KEEP clause.
- **Dictionaries:** Sets of key/value pairs for efficient look-up.
- **Vectors:** Ordered sets of values or data structures.

### Flex Operators

Flex operators are custom, programmable event handlers that go beyond what you can do with the standard relational operators of CCL. They are used to create a new derived Stream or Window, in the same way that a continuous query does. However, instead of passing the incoming event through a SELECT statement to produce the result, a Flex Operator invokes an event handler written in SPLASH upon the arrival of a new event. Thus, a user can write sophisticated custom operators in SPLASH and thus not limit processing to the set of operations that can be achieved via the standard SQL relational operators.

### Built-in Function Library

Each continuous query will include a set of “column expressions” to compute the value of each field in the new event being produced. These expressions can draw from the extensive set of built-in functions.

- **Math**
- **Statistical analysis**
- **Geometry**
- **Logical**
- **String**
- **Date/Time**
- **Calendar**
- **Aggregation**
- **Sets**
- **Bitwise**
- **XML**

### Custom Functions and Operators Defined in SPLASH

SPLASH (Stream Processing Language SHell) is a simple procedural scripting language that brings extensibility to CCL. While the declarative style of SQL is both simple and familiar, it can be restricted. Sybase ESP overcomes this with SPLASH. SPLASH scripts can be embedded in-line within a CCL project to define custom operators and functions.

### User Defined Functions in C/C++ or Java

External function libraries in C, C++, or Java can be used in ESP expressions once the library has been loaded. The interface specification is defined in the ESP CCL Programmers Guide.
Modularity

An ESP Project can invoke external CCL modules. This facilitates re-use and team-development, where different team members can be responsible for different modules. It can also be used to improve the maintainability of large complex projects.

An Project can use a CCL module defined in another CCL file by first using the IMPORT command to include the CREATE MODULE statement from the external file. It then uses the LOAD MODULE statement to invoke the module, and a single module can be invoked more than once within a project. When a module is loaded, the input and output bindings are set. Additionally, any parameters used by the module are set when the module is loaded. Thus a module that computes a moving average could have the duration set via parameter such that it could be loaded one place to produce a 10 minute moving average and loaded another place to produce a 2 hour moving average.

Database Integration

It’s often necessary to use information stored in external databases to process an event. For example, a continuous query that is processing an incoming event that contains a customer ID may need more information about that customer. This additional reference information might come from a customer database that is indexed by customer ID.

There are three ways to use data from external databases:

1. **Pre-Load:** Data from an external database is loaded into an ESP table and is held there, ready and available, for use in processing incoming events. An ESP Database Input adapter can be used to pull data from an external database. The adapter can be configured to get the data once when the project is started, or it can be configured to periodically retrieve the data. Alternatively an external process can “push” the data from the database to the ESP server at any time.

2. **As-Needed:** Data from an external database can be fetched when it is needed to process a new event. This is done via the getdata() function that is included in an expression within a continuous query. When an incoming event triggers execution of the expression containing the getdata() function, an SQL query is made against an external data source, the data is retrieved, and can then be used to compute the expression.

3. **Streaming:** The Sybase Replication Server can be used to turn transactions in external databases into event streams in real-time, streaming them into the ESP server via the ESP Replication Server Adapter.

Designed for Performance – High Throughput with Low Latency

Sybase ESP was designed from the ground up for speed and scalability, capable of meeting the performance needs of the most demanding applications. On a 2 CPU Linux server, for example, an ESP server can process well over 100,000 messages per second. Latency, measured from the time a message arrives until processing is complete and results have been produced, is typically in the range of a fraction of a millisecond to a few milliseconds. What’s more, the ESP engine has been designed to deliver consistent latency. When measuring latency, some systems may display low average latency but with a wide range of values including spikes that can run into seconds. For applications where latency is critical, ESP can be counted on to deliver a consistent latency profile.

To minimize latency, ESP uses a real-time dataflow architecture that moves the data through the continuous queries. Unlike databases that store then query, the entire ESP architecture is event-driven and designed to flow data through the system as quickly as possible. All data is held in memory, and even where windows need to be recoverable, a proprietary high speed log store mechanism is used to provide data recoverability with minimal impact on speed.

For scalability, ESP leverages a 64 bit multi-threaded architecture to fully leverage the parallel processing capabilities of multi-core/multi-cpu servers. To further extend this, the ESP cluster architecture manages any number of ESP “projects” running across all available servers with dynamic data bindings to efficiently pipe information between different projects. Projects can scale beyond the boundaries of a single machine by splitting into multiple interconnected sub-projects, each running on a different server in the cluster.
**Designed for Versatility**

As described earlier in this paper, different applications have different needs. Many CEP products are designed to address a single type of application. For example, there are a number of CEP “rules engines” that are designed expressly for situation detection. That’s fine if all you need is situation detection, but the technology may not be extensible to other types of applications. Sybase ESP was designed to address the widest possible range of event processing requirements:

- Monitor incoming data streams for conditions that represent opportunities or threats
- Augment data streams with data from other sources and/or computed values
- Group data by different dimensions, producing high level summary data or statistics
- Consolidate data from multiple heterogeneous systems, forming a single aggregate view or stream
- Operate on large data sets spanning large time windows
- Collect raw and/or result data for use in historical analysis, reporting, or to provide an audit trail

Some of the specific aspects of the ESP architecture that give it this versatility include:

- **Native state management:** Incoming messages can be processed as inserts, updates, deletes or upserts. This lets ESP efficiently manage data windows where incoming messages don’t just represent a new data point in a time series, but represent an update to previous information. Many CEP implementations don’t handle updates and deletes — they treat all incoming messages as new data points in a time series. The reality is that many data streams produce updates, changes, and cancellations. Whether it’s changes to an order book, or a correction to data previously sent, these updates need to be applied to previously received data to maintain an accurate view of the current state.

- **SQL + SPLASH for extensibility:** CCL, the Sybase event processing language that is derived from SQL, provides familiarity and ease of use. The inline SPLASH scripting language provides extensibility, making it easy to add custom operators and functions with the fine-grained control of a procedural scripting language with data structures for efficiently maintaining state between events.

- **On-demand queries:** An interface that allows all event windows to be queried by external applications as if they were held in a database. The ODBC/JDBC interfaces provide support for off-the-shelf query applications and allow for snapshots of current data sets.

- **Rich Subscriptions:** Each subscription to an ESP output stream can be tailored to the needs of the consuming application. Subscriptions to windows can deliver the full window followed by updates, or just updates. Subscriptions can include an SQL SELECT statement to further filter, aggregate or otherwise refine the data.

- **Security:** Built-in security including access control, authentication and encryption.

- **Database Integration:** ESP has strong integration with databases both to capture event data and to load context from databases for use in processing events. And Sybase replication technology can be used for real-time change capture on databases, turning database transactions into real-time event streams.

**A Distinguished Heritage**

Sybase Event Stream Processor 5.0 represents the latest innovations in complex event processing, building on the success of both the Aleri Streaming Platform and the Coral8 Engine. In 2009 Sybase determined that CEP would play a strong role in the future of real-time analytics and launched “Sybase CEP” which was an updated version of the Coral8 product that had established a strong industry following and was recognized as one of the leading CEP products in the market.

In 2010 Sybase saw the opportunity to further strengthen its CEP offering by acquiring both the Coral8 and Aleri product lines. Aleri had also established a strong position in the industry, built a reputation for high performance, and was recognized by Forrester as a “standout leader” in the field.

Following the acquisition of these products, Sybase set out to build a next generation CEP platform that combines the best of these two products. The result is the Sybase Event Stream Processor (ESP) that brings the richness of the Coral8 “CCL” event processing language together with the performance and scalability of the Aleri execution engine. The new ESP studio builds on the successful Aleri Studio by combining both visual dataflow editing with textual CCL editing in a single, seamless environment — a first in the industry.
What’s New in 5.0

Sybase Event Stream Processor 5.0 builds on the strengths of the Aleri Streaming Platform 3.2 by introducing the following new features:

• CCL

CCL – short for “continuous computation language” is an SQL-based language for defining continuous queries that are applied to event streams. It replaces AleriML, the XML-based modeling language that was used in all Aleri releases up through 3.2. It has the advantages of:
– More human friendly – escapes the clutter of xml tags
– More flexible – can express complex queries that aren’t limited to the AleriML primitives
– Easy to learn, since it’s based on SQL

CCL originated in the Coral8 product and is now the language of Sybase CEP R4, the latest version of Coral8. It was the most popular aspect of Coral8, thus the adoption in ESP. Note, however, that ESP will be using “CCL 2”. We have simplified and streamlined the language, so while it carries forth the style and concepts of the original CCL, some things have changed.

Note that SPLASH remains unchanged. The move to CCL doesn’t affect SPLASH, and SPLASH can still be embedded in CCL to define custom functions and custom operators (Flex Operators – formerly known as Flex Streams).

• Studio Support for both Visual and Textual Editing

With the introduction of CCL, there is also a fully functional CCL editor in the new ESP Studio. At the same time, the ESP studio continues to feature a Visual Editor, similar to the editor in the Aleri studio.

What’s particularly noteworthy is that users don’t have to choose: they can switch between the visual editor and the CCL editor at will. Changes made in one, will be reflected in the other. For new users, the visual editor is a great starting point, eliminating the need to learn CCL syntax. For experienced programmers, who are comfortable working with CCL, the CCL editor provides a highly efficient way of working. And even CCL power users will value the ease of navigating large programs that the visual editor provides.

• Advanced Cluster Architecture

While Aleri 3.x was limited to a single live project per server, ESP introduces a new clustering architecture that allows any number of projects to be run on a single ESP “cloud” that can span any number of machines. Projects can be started and stopped dynamically and independently of all other projects, and can dynamically bind to streams and windows in other project. For more information see the section on clustering later in this paper.

• Add New Continuous Queries on the Fly

One of the benefits of the new cluster architecture is that it supports very dynamic environments. New continuous queries can be added to a running server at any time in the form of new Projects. Projects that are no longer needed can be stopped at any time without affecting other projects (downstream projects notwithstanding of course).

Let’s take an example: a user wants to perform a new aggregation on an existing output stream in a live project. The user can create a new project that subscribes to the existing output stream, and applies the desired aggregation (plus filtering and any other computations), and then add this project to the server. It immediately becomes live, subscribing to the other project, and will run until stopped.

• Modularity

New modularity features facilitate re-use and team development, as well as making it easier to maintain large complex projects.
– Modules can be defined using a CREATE MODULE statement
– Modules can contain streams and windows, which can contain continuous queries
– An IMPORT statement (functionally similar to #include in C/C++) lets you reference declarations, modules and other elements defined in separate .ccl files
– Modules can be parameterized, where the parameters are set when the module is loaded
– A LOAD MODULE statement invokes a module, sets the parameters, and defines the bindings to streams/windows that supply data to the module and receive data from the module
– The same module can be loaded multiple times in a single project
• New Data Types
ESP adds the following new data types:

- BigDateTime – a timestamp with microsecond granularity
- Interval – microsecond granularity
- Binary
- Boolean

It also enhances the MONEY data type to allow the precision to be set at use, rather than set globally.

• New Pub/Sub API
ESP R5 introduces a new pub/sub API that is designed for improved performance as well as being simpler to use. Unfortunately it does mean that any custom integration done using the Aleri 3.x pub/sub API will need to be updated to use the new API. Some features of the new API include:

- Higher performance
- Simpler – fewer steps to set up a subscription and start publishing or subscribing
- The C++ API as been replaced by a C API to avoid compiler dependencies, allowing you to use the compiler of your choice
- A choice of programming models for receiving events:
  - Synchronous
  - Call back
  - Select method
- Threading options

Full details can be found in the ESP API specification.

• Adapters
The following new adapters will be included in the base product:

- MSMQ
- Websphere MQ

The following new licensable (optional) adapters will be available with ESP:

- ESP Adapter for ActivFinancial ActivFeed
- ESP Adapter for Interactive Data PlusFeed/PlusBook

Additional enhancements in the area of interfaces, adapters and SDKs include:

- The database adapter will support any ODBC driver
- The JMS adapter will support any JMS provider
- Most adapters will be available on all supported platforms

• Studio Run-Test Environment: Manage Multiple Servers, Multiple Projects
The Studio Run-Test Environment has been restructured with a navigation tool that let’s you start/stop projects on multiple ESP servers and run the test tools against any live project.

• Analytics
The library of built-in functions has been significantly expanded with over 70 new functions. These include:

Numeric: atan2, compare, cosd, cosh, distance, distancesquared, log2, pi, random, sign, sind, sinh, tand, tanh

Bitwise operations: bitand, bitclear, bitflag, bitmask, bitnot, bitor, bitset, bitshiftleft, bitshiftright, bittest, bittoggle, bitxor

Sets: avgof, maxof, minof

Aggregation: corr, corrobias, covar_pop, covar_samp, exp_weighted_avg, meandeviation, median, regr_avgx, regr_avgy, regr_count, regr_intercept, regr_r2, regr_slope, regr_sxx, regr_sxy, regr_syy, var_pop, var_samp, vwap

String functions: ascii, char, left, regexp_firstsearch, regexp_replace, regexp_search, trim, upper

Date and time: dateceiling, datefloor, daterround, dayofmonth, dayofweek, dayofyear, timeToUsec, hour, microsecond, minute, month, second, usecToTime, to_timestamp, to_interval, year

Binary: extract, concat, length, to_binary, hex_binary, base64_binary, to_string, string, hex_string, base64_string, tonetbinary, fromnetbinary

Column access: get*columnbyname(record, colname), get*columnbyindex(record, colindex), isInsert, isDelete
• Other – Misc. Enhancements
  – Project configuration data can be kept separate from the business logic: all physical deployment configuration elements (host names, etc) are defined in project configuration files that are separate from the CCL that defines the business logic. This makes it easier to move Projects from a test environment to a production environment, or to deploy a Project in multiple locations.
  – Uses SySAM licensing mechanism. Flexible for both local license and license server deployment. 30 day run-time grace period
  – Internationalization: ESP R5 meets all Sybase internationalization standards, allowing it to be localized, including support for non-English versions. All features are UTF-8 compliant.

PART OF A COMPREHENSIVE REAL-TIME ANALYTICS PLATFORM

Sybase ESP is part of a family of products providing a complete real-time analytics platform. Sybase IQ and Sybase RAP bring data persistence to ESP, providing the ability to capture and store massive amounts of event data in a high performance analytics server that can scale to handle the largest data sets, support simultaneous queries by many users, in an architecture that is optimized for analytics.

Integrated platform for analysis of streaming, real-time and historical market data

Sybase RAP and Sybase ESP consolidate streaming, real-time and deep historical data into one integrated platform that ensures data for quantitative modeling, high-velocity trading and risk management applications is available consistently, enterprise-wide, as one common view of the truth. Sybase RAP and Sybase ESP both leverage SQL as the standard query language. With SQL, developers use existing skill sets resulting in lower ongoing maintenance costs.

Historical repository provides cost-efficient storage and dramatically faster analytics

Sybase IQ, the historical data repository and analytics engine within Sybase RAP, efficiently manages vast amounts of data — petabytes and beyond — with unique patented compression capabilities that reduce the size of raw input data by up to 50–70 percent. With its advanced compression, firms can afford to capture and store all event data, or they can use Sybase ESP to perform “smart capture” — only capturing meaningful events. Extensive support for parallel loads ensures that data from any source or location is quickly available. The patented indexing and the columnar architecture enable complex analyses to be performed 10 to 100 times faster than traditional transactional databases. With these capabilities — compression, fast loads, and dramatically faster queries — trading desks can perform time-critical analyses to predict price movements and identify arbitrage opportunities. Quantitative analysts can design, develop and run powerful, complex models against longer term trend data for competitive advantage.

Time series and windowing analysis support

Sybase RAP provides built-in functions for analyzing and manipulating time series data. Time series analysis requires the ability to identify and potentially remove outliers, to fill in missing values, and to factor in seasonality and other trends. These and other critical functions — such as ARMA (auto-regressive moving average), Box-Cox transformations, measuring correlation and covariance, and lack of fit test for time series data — are provided via a set of built-in functions in RAP. When combined with IQ’s OLAP windowing features, quantitative analysts can achieve powerful insight into market dynamics.

The rich set of time series and other analytic SQL built-in functions available out-of-the-box accelerates implementation of key capital markets applications, without limiting flexibility.

Extensible enterprise analytics

Customers extend RAP’s built-in analytics with their own custom or preferred third party analytics by leveraging its user defined functions (UDFs). This key feature permits analytic libraries written in C or C++ to be loaded into a running RAP server and then exposed as functions used in queries or other SQL statements. UDFs allow custom analytics to be applied to analysis of massive amounts of data along different dimensions to compute result sets and multidimensional cubes that exploit the company’s custom or preferred third party analytics libraries.
RAP analytics can also be extended and leveraged using R, the popular programming language and environment for statistical computing and graphics. R programs can be called from IQ as SQL functions, returning results directly to clients or using the results for additional server-side processing. Additionally, a quant or statistician can reach out from the R environment and access data managed within RAP. This use with R pairs the analytical and graphical power of R with the powerful data management of RAP.

Today, quant teams face the challenge of ensuring that their algorithms, and the underlying assumptions on which they are based, are leveraged consistently by other team members within a large enterprise. With Sybase RAP, analytics are shared as SQL functions, ensuring access to the analytics is secured according to a firm’s standards. This permits calculations that identify and remove outliers, for example, to be standardized and applied consistently within and across teams. With the ability to share both a common view of enterprise-wide data and to reuse a common set of functions, RAP increases quant team productivity and analytics quality.

*Real-time in-memory cache stores intra-day market data at high speed and with low latency*

Sybase RAP includes an in-memory cache for immediate on-demand access to data that needs to be readily available in support of transactional applications. Events can be captured in the RAP cache at high speed and with low latency.

*Enterprise-class scalability*

Sybase offers the only event analytics platform flexible and powerful enough to support enterprise-wide analytics requirements, from streaming and real-time continuous analytics to multi-year historical analyses. The platform supports incremental scale out of virtually any workload upon clusters of commodity hardware.

*Sybase ESP+RAP in the Capital Markets*

The combination of ESP and RAP is often used in the capital markets for tick capture and trade capture in support of strategy development, live trading, and compliance.

This powerful combination provides for high speed data capture, the ability to establish “smart capture” rules to determine what data to save, and an extremely scalable analytics store with built-in time series analysis functions.

Unlike other tick capture solutions, Sybase RAP can be deployed as an enterprise data platform, able to handle large numbers of simultaneous queries from a large user population.