SYBASE®
An SAP® Company

New Features Summary

Sybase Event Stream Processor 5.1 SP01

DOCUMENT ID: DC01616-01-0511-01
LAST REVISED: November 2012

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

Contents

New Features

This New Features Summary provides high-level, overview information on new features introduced in Sybase® Event Stream Processor 5.1. This is a cumulative summary of all features introduced since the initial release of 5.1 up to and including the current Support Package. Features are listed with the particular release in which they were first introduced, and remain available in all subsequent versions unless otherwise specified.

New Features in 5.1 SP01

Sybase Event Stream Processor 5.1 SP 01 introduces a single, but significant, new feature: a high speed output adapter for SAP® HANA.

The new SAP HANA output adapter can use multiple parallel ODBC connections to load data rapidly from Event Stream Processor into an SAP HANA database. The adapter is designed for high-volume, high-speed processing to take full advantage of the SAP HANA real-time, in-memory architecture.

The introduction of the SAP HANA output adapter lets you integrate Event Stream Processor into new or existing SAP HANA environments with virtually no configuration on the SAP HANA side.

This adapter also publishes statistics to the _ESP_Adapter_Statistics metadata stream. Subscribe to this metadata stream to view statistics specific to the SAP HANA adapter.

New Features in 5.1

Sybase Event Stream Processor 5.1 introduces new features, new adapters, a new management and administration tool, and performance enhancements.

CCL

Continuous Computation Language (CCL) is the primary event processing language of Sybase® Event Stream Processor.

Sybase Event Stream Processor 5.0 introduced a simplified, streamlined version of CCL based on the CCL offering in Sybase CEP R4. Version 5.1 introduces the following additional CCL functionality:

Jumping Windows
Using the **KEEP** clause in CCL, you can define a retention policy for a Named or Unnamed Window. Window retention policies include time-based policies (the time duration for which

a window retains rows) and count-based policies (the maximum number of rows that the window can retain). In version 5.0, retention policies always created sliding windows. In a sliding window, individual rows are deleted once a maximum age is reached or the maximum number of rows are retained (depending on the policy type).

Version 5.1 introduces the **EVERY** modifier to the **KEEP** clause. When you use the **EVERY** modifier in your **KEEP** clause, you create a retention policy that uses a jumping window. In a jumping window, all of the retained rows are deleted when the time interval expires or a row arrives that would exceed the maximum number of rows.

For example, if you create a retention policy to keep five rows of data in a sliding window, the window holds the first five rows as they feed into the stream. When the sixth row enters the stream, the first row slides out of the window, and the sixth row slides in. If you define a retention policy to keep five rows of data in a jumping window, the window holds the first five rows as they feed into the stream. When the sixth row enters the stream, all five retained rows jump out of the window, and the sixth row jumps in.

The following sample code, provided only for illustrative purposes, creates an unnamed jumping window on the source Source1 that will retain every row until the end of a 100 second interval.

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

For detailed information on syntax and usage, see the *Programmers Reference*.

Content-based Retention

When working with the **KEEP** clause in version 5.1, you can now use the **PER** sub-clause to retain data based on column content. In a time-based policy, data is retained for a specific time period (specified by **interval**) for each unique column/value combination. In a count-based policy, the specified number of rows (specified by **count**) are retained for each unique column/value combination. In either case, you can specify one or several columns, but you cannot use the same column more than once in the same **PER** sub-clause.

In the following sample, provided only for illustrative purposes, Win1 is a derived window that retains data for each unique value combination of Col1 and Col2. The window is a Jumping window that retains 5 rows. Upon arrival of the sixth unique value combination of Col1 and Col2, the first 5 rows are deleted and the sixth row is stored.

```
 CREATE WINDOW Win1
PRIMARY KEY DEDUCED
AS
SELECT * FROM InputWin
KEEP EVERY 5 ROWS PER (Col1, Col2);
```

For detailed information on syntax and usage, see the *Programmers Reference*.

Splitting Input into Multiple Outputs

The **CREATE SPLITTER** statement lets you evaluate an input stream for particular conditions. As the rows in the stream are evaluated, the splitter directs them to different target streams based on which conditions are met.

For example, you can define a splitter statement that evaluates stock symbols and directs them to different streams depending on whether the symbols are for software vendors, hardware vendors, or other vendors. In the following sample code, provided for illustrative purposes only, the splitter evaluates the source stream Trades. When the Symbol column for a row contains "IBM" or "ORCL" the row is added to two different target streams, ProcessHardWareStock and ProcessSoftwareStock. When the Symbol column for a row contains "SAP" or "MSFT" the row is added to the target stream ProcessSoftwareStock. All other rows are sent to the target ProcessOtherStock.

```
CREATE SPLITTER Splitter1 AS
WHEN Trades.Symbol IN ('IBM', 'ORCL' ) THEN ProcessHardWareStock,
ProcessSoftwareStock
WHEN Trades.Symbol IN ('SAP', 'MSFT') THEN ProcessSoftwareStock
ELSE ProcessOtherStock
SELECT * FROM Trades;
```

When a stream is filtered into more than two target streams, a splitter is typically more efficient, both in terms of CPU utilization and throughput, than an equivalent construct composed of two or more streams that does a filter. Unlike other streams in Event Stream Processor, a splitter and all its target streams run in a single thread.

For detailed information on syntax and usage, see the *Programmers Reference*.

Automatic Key Generation

When you are working with input data that does not have a natural primary key, use the **AUTOGENERATE** clause to automatically generate values that can be used, for example, as a primary key for the data. Specify a column that uses the `long` data type. At runtime, values for that column are generated automatically for each insert in the stream. By default, values start at zero, but you can override this using the FROM clause and specifying a start value with either a parameter or an explicit long number.

The automatically-generated column is incremented only on an insert, and any preexisting value in an automatically-generated column is, on insert, ignored. On an update, delete, or upsert, a preexisting value in the automatically-generated column is used as-provided in the input row. This rule has the potential to produce duplicate rows in a window, so do not use the **AUTOGENERATE** clause with upserts, especially when the automatically generated column is a primary key.

The following sample code, provided for illustrative purposes only, creates an input stream named Trades with a column named TradeId for which the values are automatically generated.

```
CREATE INPUT STREAM Trades
SCHEMA (TradeId long, Symbol string, Shares integer, Price money(4))
AUTOGENERATE (TradeId);
```

For detailed information on syntax and usage, see the *Programmers Reference*.

ESP Studio

Event Stream Processor Studio provides visual and textual authoring capabilities as well as a run-test environment. This enables you to develop, test, and configure projects in a common interface.

The CCL textual editor lets you edit CCL files as text, and provides syntax completion options, syntax checking, and error validation. Experienced programmers can work in the CCL editor exclusively, or use it as a supplement to the visual editor.

The visual editor lets you create and edit projects without learning CCL syntax. The visual editor provides support for all CCL functions, including the new jumping windows, splitter, and automatic key generation features introduced to CCL in version 5.1.

To support the CCL modularity feature, the Studio can be configured with a repository for blocks of CCL code called modules that can be used in multiple projects.

For detailed information on using the Studio, including information on configuring jumping windows, splitter, and automatic key generation in the visual editor, see the *Studio Users Guide*.

Project Configuration

Event Stream Processor project configuration separates physical deployment elements from business logic.

Event Stream Processor 5.1 includes a new project configuration option for handling bad records.

To save bad records to a file, select the bad-record-file option for **Project Type**, and indicate the file name of an ESP project for **Value Field**. If a file name is not specified, bad records are discarded. See the *Administrators Guide*.

Monitor and Administer Event Stream Processor

Event Stream Processor includes an operational console, Sybase Control Center (SCC), that you can use to monitor and administer ESP nodes.

You can use this tool to monitor performance of individual nodes or clusters. It provides statistics for:

- Projects
- Streams
- Connections
- Adapters
- Publishers
- Subscribers

You can also use SCC to:

- Start and stop nodes, projects, and adapters
- Configure alerts on various statistics
- View the file activity report for the Sybase IQ Output Adapter
- View ESP project and cluster node log files
- View a graphical representation (topology) of the ESP cluster

For information on using SCC, see the *Sybase Control Center for Event Stream Processor* online help.

Adapters

In addition to the full set of adapters provided in version 5.0, Event Stream Processor 5.1 includes support for two new adapters.

- The new Adaptive Server Enterprise (ASE) Output adapter reads data from an Event Stream Processor stream and writes it to the Adaptive Server Enterprise using the high speed Open Client Bulk Loading API. This adapter also publishes statistics to the new _ESP_Adapter_Statistics metadata stream. Subscribe to this metadata stream to view statistics specific to the ASE adapter.
- The new Sybase IQ Output adapter reads data from Event Stream Processor and writes it to Sybase IQ load files in the native Sybase IQ binary format. This adapter also publishes statistics to the new _ESP_Adapter_Statistics metadata stream. Subscribe to this metadata stream to view statistics specific to the Sybase IQ adapter.
- **Note:** The new Sybase IQ adapter replaces the adapter shipped in version 5.0, including the **esp_iqloader** utility. To continue using the 5.0 version of the Sybase IQ adapter, do not install Event Stream Processor 5.1 in the same directory as 5.0.

The generic Database input and output adapters, db_in and db_out, have also been updated:

- For ODBC connections, performance for batch inserts is enhanced.
- For JDBC connections, the generic JDBC driver is improved for better connectivity with some databases.

For a full list of supported adapters, see *Adapters Supported by Event Stream Processor > Adapter Summary* in the *Adapters Guide*.

Reporting Infrastructure

Event Stream Processor infrastructure enhancements make it possible to read adapter statistics, adapter latency, CPU usage and more from metadata streams. SDK enhancements also make it possible to determine TCP packet sizes to determine how much room is available when sending messages from the SDK.

Metadata Stream Enhancements

In version 5.1, the new metadata stream _ESP_Adapter_Statistics makes it possible for both internal and external adapters to report custom statistics. To read the statistics, subscribe to this metadata stream using standard SDK subscription calls.

The _ESP_Connectors metadata stream now includes an additional column, latency, through which an internal adapter can report on any latency it introduces. This latency information is then reported through SCC.

The _ESP_Project_Monitor metadata stream reports on CPU utilization, memory consumption, and number of threads on a per-project basis. As with other monitor-related streams such as _ESP_Streams_Monitor and _ESP_Clients_Monitor, this metadata stream is only updated when the time granularity option is set to a value greater than zero.

The _ESP_Streams_Monitor and _ESP_Clients_Monitor metadata streams have been extended to report on system and user CPU percentage and CPU time.

For information on metadata streams, see the *Administrators Guide*.

SDK Support for Network Parameters

The Java, C/C++, and .NET SDKs now provide a new API that lets you determine how much room there is in the TCP packet when sending from the SDK. Information provided includes the MTU size, IP header size, TCP header size, and ESP header size.

For a connected publisher (single socket connection), the SDK now retrieves:

- ESP binary data block header size (all SDKs, all platforms).
- TCP Maximum Segment Size (MSS) for the connected socket. The MSS is computed by the system as MTU – IP Header Size – TCP header size and denotes the space available for data in a single packet that can be sent without IP fragmentation (C/C++ SDK, Linux and Solaris only).
- Maximum transmission unit (MTU) of the connected interface and default IP and TCP header sizes (Java SDK, Linux and Solaris only).

For a message writer (the component that corresponds to one particular stream on the engine), the SDK now retrieves the row header size for the stream.

Security

Security in Event Stream Processor is improved by requiring authentication for all connections to clusters.

Users connecting to a cluster from Sybase Control Center, ESP Studio, command-line utilities, or the SDK, must authenticate using any of the existing methods: Kerberos, LDAP, RSA, or by specifying user name/password credentials (native OS).

Users connecting to the local cluster in Studio authenticate as the studio user, with a user-defined password that is valid for the current session only.

PowerDesigner Extensions for Event Stream Processor

Sybase PowerDesigner® is a tool for creating and manipulating schema definitions which can be used within Sybase Event Stream Processor. Integrated modeling supports efficient schema definition and database design, and consistent production deployments.

PowerDesigner extensions are used for customizing and extending PowerDesigner metaclasses, parameters, and generation. With the extended model setup, you can create and modify schema definitions in the ESP schema model, convert Sybase IQ and Sybase Adaptive Server Enterprise physical models into ESP schema logical models, and import and export schema definitions to and from CCL files. You can also use PowerDesigner for impact and lineage analysis.

PowerDesigner extensions can only be used on Windows platforms. You must purchase PowerDesigner separately to use this feature.

The PowerDesigner extensions installed with Event Stream Processor provide a sample project which includes:

- A sample ESP schema logical model
- Sybase IQ and ASE physical data models

You can use the sample project to modify or extend existing models, or as a guideline for creating new models within the ESP model categories.

In addition, PowerDesigner can be used to produce a set of data definition language (DDL) statements directly from the physical data models. These DDL scripts can be used to create database objects for your Sybase IQ and ASE databases.

For more information on using PowerDesigner, see *PowerDesigner for Event Stream Processor* in the *Programmers Guide*.