SYBASE®

Utilities Guide

Sybase Aleri Streaming Platform 3.1

DOCUMENT ID: DC01289-01-0311-01

LAST REVISED: June, 2010

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

Table of Contents

About This Guide

1. Preface

This guide explains how to use the various scripts, utilities and tools that comprise the Sybase Aleri Streaming Platform.

2. Audience

This guide is intended for system administrators who will manage the Sybase Aleri Streaming Platform along with the applications built for it. It includes all of the UNIX® style manual pages for command line utilities.

3. Organization

Chapter 1, *Overview of Sybase Aleri Streaming Platform Executables* briefly describes the command-line executables that are available for managing the Sybase Aleri Streaming Platform.

Chapter 2, *AleriRT* describes the Real-Time Data add-in for Microsoft Excel®.

Chapter 3, *Server Executables* contains man pages for those executables used in starting, monitoring, and querying the Streaming Processor.

Chapter 4, *Command and Control Executables* contains man pages for those executables used in controlling and querying information from a running Streaming Processor.

Chapter 5, *Publish and Subscribe Executables* contains man pages for those executables used in publishing records to and subscribing to records from the Streaming Processor.

Chapter 6, *Authoring Executables* contains man pages for those executables used in authoring models.

Note:

The man pages for the scripts, tools and example code are included in the package. To make these man pages accessible from the command line via the **man** command, add the path where they were installed to the `MANPATH` environment variable. Enter

```
export MANPATH=$MANPATH:$PLATFORM_HOME/man
```

where `$PLATFORM_HOME` is the environment variable containing the location of the Sybase Aleri Streaming Platform that you created in the installation procedure.

4. Related Documents

This guide is part of a set. The following list briefly describes each document in the set.

Product Overview	Introduces the Aleri Streaming Platform and related Aleri products.
Getting Started - the Aleri Studio	Provides the necessary information to start using the Aleri Studio for defining data models.
Release Bulletin	Describes the features, known issues and limitations of the latest Aleri Streaming Platform release.
Installation Guide	Provides instructions for installing and configuring the Streaming Processor and Aleri Studio, which collectively are called the Aleri Streaming Platform.

Authoring Guide

Provides detailed information about creating a data model in the Aleri Studio. Since this is a comprehensive guide, you should read the *Introduction to Data Modeling and the Aleri Studio*. first.

Authoring Reference

Provides detailed information about creating a data model for the Aleri Streaming Platform.

Guide to Programming Interfaces

Provides instructions and reference information for developers who want to use Aleri programming interfaces to create their own applications to work with the Aleri Streaming Platform.

These interfaces include:

- the Publish/Subscribe (Pub/Sub) Application Programming Interface (API) for Java
- the Pub/Sub API for C++
- the Pub/Sub API for .NET
- a proprietary Command & Control interface
- an on-demand SQL query interface

Utilities Guide

Collects usage information (similar to UNIX® man pages) for all Aleri Streaming Platform command line tools.

Administrators Guide

Provides instructions for specific administrative tasks related to the Aleri Streaming Platform.

Introduction to Data Modeling and the Aleri Studio

Walks you through the process of building and testing an Aleri data model using the Aleri Studio.

SPLASH Tutorial

Introduces the SPLASH programming language and illustrates its capabilities through a series of examples.

Frequently Asked Questions

Answers some frequently asked questions about the Aleri Streaming Platform.

Chapter 1. Overview of Sybase Aleri Streaming Platform Executables

1.1. Streaming Processor Executables

sp The debug version of the Streaming Processor executable.

sp_clustermgr The Cluster Manager is an interactive command-line interface that supports simple commands to start, stop and inquire about the status of a distributed model (cluster).

sp_clustermon The Cluster Monitor monitors the health of a running cluster and repairs failed modules.

sp_ld The Launch Daemon executes commands used to manage the modules of a cluster under the direction of the **sp_clustermgr** and **sp_clustermon** commands.

sp_monitor The Monitoring Tool receives and displays performance data from a running instance of the Sybase Aleri Streaming Platform. The performance data is written to the standard output.

sp_query The Query utility reads an SQL query from the standard input and forwards it to a running instance of the Sybase Aleri Streaming Platform for execution. The result of the query is written to the standard output.

sp-opt The optimized version of the Streaming Processor executable, which is suitable for production use.

sp_server The Server command initiates the Streaming Processor (**sp-opt**) and the encryption proxy server for the Command and Control interface.

sslwrap The secure communication proxy offers support for secure communications with the Command and Control interface. (Presently, the Abyss (HTTP) server embedded in the Sybase Aleri Streaming Platform for the command and control interface does not natively support encryption.)

1.2. Command and Control Executables

sp_cli The interactive Command-Line shell can query meta-information, inject rows of data, make table snapshots, and control a running Streaming Processor instance. It can also be used to stop the Streaming Processor, fetch the list of streams and their definitions, and determine the host and the port of the Gateway interface.

sp_cnc The Command and Control utility executes individual control commands. It is supplied with the source, to serve as an example of writing client applications.

1.3. Publish and Subscribe Executables

sp_convert The Converter utility reads XML or delimited records from standard input and produces binary format records on standard output.

sp_histexport The History Export utility moves data from a running instance of the Streaming Processor to a historical vector store that is part of an OLAP server. The data to be transferred is described as an XML file which is passed as input to the **sp_histexport** executable.

sp_subscribe The Subscribe command connects to a running instance of the Streaming Processor and subscribes to streaming data. The received records are converted to XML (or optionally delimited format) and written to standard output.

sp_upload The Upload command reads binary records from standard input and publishes them to a current instance of the Sybase Aleri Streaming Platform through the Gateway interface.

sp_archive The Archive command archives data from the specified stream(s).

sp_viewer The Streamviewer enables users to view in a graphical user interface (GUI) the streams that are being maintained on a specified Sybase Aleri Streaming Platform instance.

1.4. Authoring Executables

sp_sql2xml The SQL to XML Translator converts a set of Aleri SQL statements to the corresponding AleriML representation.

sp_studio This shell script launches the Aleri Studio, which can be used to visually author data models, and to initiate and monitor the Streaming Processor.

Chapter 2. AleriRT

AleriRT is a real-time data add-in for Microsoft Excel® that lets you view records in one or more instances of the Sybase Aleri Streaming Platform and publish records to it. It is not packaged with the Sybase Aleri Streaming Platform; for information about downloading and installing it, contact your Sybase representative.

On the display side, AleriRT lets you select the streams and view the columns within the stream. It also enables you to filter records based on data values. You can view the most recent “N” records, or the most recent “N” records that match a specified filter (where “N” is the specified number of records).

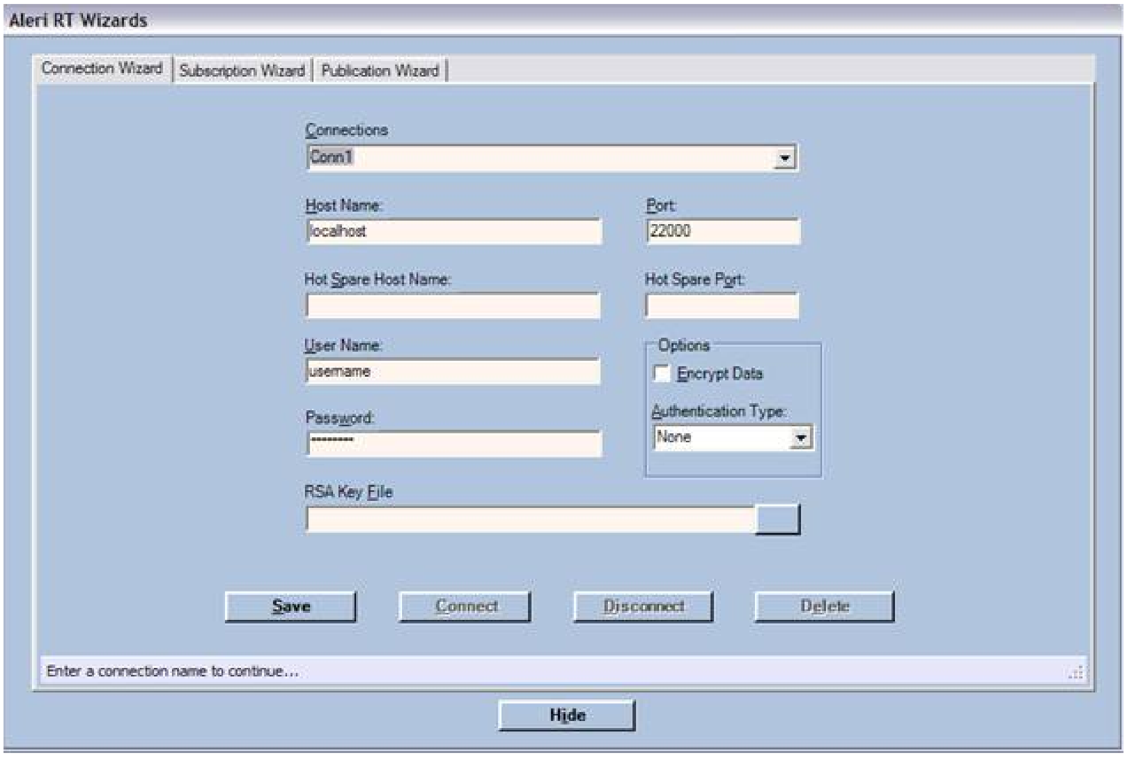
On the publish side, AleriRT provides the ability to automatically publish data whenever data changes in a range of cells. You can also manually publish data to the Sybase Aleri Streaming Platform by selecting a range of cells and using the Publish Wizard.

2.1. Using AleriRT

Click on the **AleriRT** button to bring up the AleriRT Wizard screen. The screen contains tab pages for three wizards: the Connection Wizard, Subscription Wizard, and Publish Wizard. This screen is non-modal, meaning that the Excel worksheet can be modified even when the screen is open.

2.1.1. The Connection Wizard

The image below shows the AleriRT Wizards screen with the Connection Wizard tab active.



Connections: Conn1

Host Name: localhost

Port: 22000

Hot Spare Host Name:

Hot Spare Port:

User Name: username

Password: ********

Options

□ Encrypt Data

Authentication Type: None

RSA Key File

Enter a connection name to continue...

The Connection Wizard provides a way to simultaneously connect to one or more instances of the Sybase Aleri Streaming Platform. The second tab is the Subscription Wizard. which lets you define and control one or more subscriptions, the results of which are displayed in Excel. The last tab is the Publication Wizard, which provides a way to manually publish data to a stream in any instance of the Sybase Aleri Streaming Platform. Each of these wizards is explained in detail below.

The Connection Wizard pane allows simultaneous connections to multiple instances of the Sybase Aleri

Streaming Platform.

Connection Enter the name of a new connection, or select from a list of previously defined connections. When a connection is selected from this listbox, all the information associated with the connection is displayed.

Host Name Enter the name of the computer on which the Sybase Aleri Streaming Platform is running. This property is required.

Port Enter the port on which the Sybase Aleri Streaming Platform is listening. Contact the system administrator to obtain this required value.

Hot Spare Host Name If you are running a hot spare Sybase Aleri Streaming Platform on another system, enter the name of that system. When the primary server fails, data is automatically retrieved from the hot spare.

Hot Spare Port Enter the port on which the hot spare is listening. This is only required when using Hot Spares.

User Name Enter the user name to connect to the host machine. This property is required unless you selected **None** as the authentication type.

Password Enter the password associated with the user name. This is an optional property that needs to be provided with PAM authentication and the underlying authentication mechanism has user names and passwords.

Encrypt Traffic If the Sybase Aleri Streaming Platform is started in encrypted mode, select this option in order to "talk" to it. Otherwise, a connection error occurs.

Authentication Type An authentication type needs to be selected from this list. The authentication type selected must match the mode that was used when starting the Sybase Aleri Streaming Platform.

RSA Key File If RSA authentication is selected, enter the RSA key file. You can either type the location and name of the key file or click on the button next to the field to browse and choose the file.

Save This button saves the connection information in a hidden worksheet associated with the active workbook. This enables the connection information to be retrieved and displayed automatically when the active workbook is reopened at a later time.

Connect After providing all the information, click this button to connect to the server. If the connection is successful, only the **Disconnect** button is available for this connection. This also causes the connection information to be saved for future use.

Disconnect Click on this button to drop the connection to the Sybase Aleri Streaming Platform. On a successful disconnect, this button is disabled and the **Connect** and **Delete** buttons are enabled. If there are queries actively using this connection then the queries are stopped after user confirmation before disconnecting.

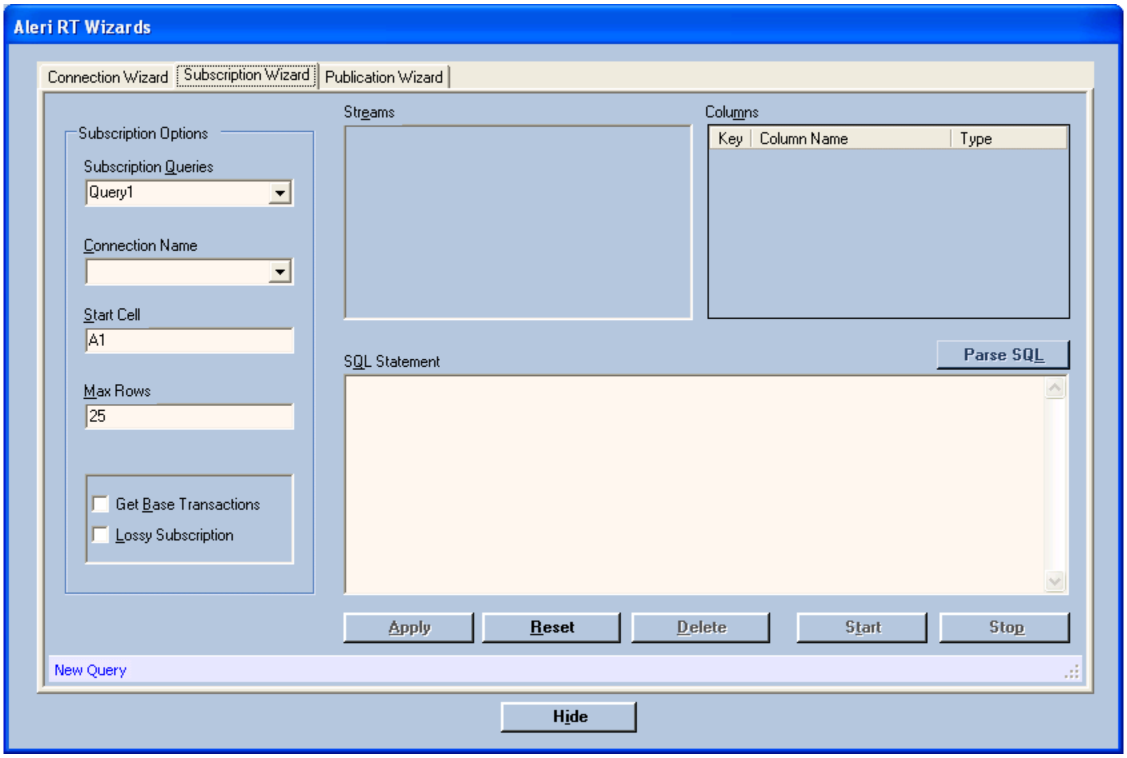
Delete Click on this button to delete a connection.

Hide Click on this button to hide the window while preserving all information. To redisplay the screen, click the **AleriRT** button on the toolbar.

The connection information for an Excel workbook is saved with the workbook.

2.1.2. The Subscription Wizard

The Subscription Wizard pane enables you to define the queries for the results to be displayed in the Excel worksheet. The figure below shows the Subscription Wizard pane.



Subscription Queries: Query1

Connection Name:

Start Cell: A1

Max Rows: 25

□ Get Base Transactions

□ Lossy Subscription

New Query

This pane has the following components:

Subscription Queries

Enter the name of a new query, or choose a previously defined query by selecting the **Query Name** from the drop down list. When a previously saved query is selected, all information associated with the selected query is automatically displayed.

Connection Name

Select the connection associated with the query from the drop down list of currently active connections. The query can only be run if the associated Connection is active.

Start Cell

Enter the location in the Excel worksheet to start inserting the real-time data formulas. Specify the location in the “A1” notation. For example, entering a value of B5 will tell AleriRT to insert the formulas as a grid starting at column B, row 5.

Max Rows

Enter the number of records (the maximum value allowed is 65536) to be displayed in the Excel worksheet. When there are more rows to be displayed than the number specified, the oldest records is discarded.

Get Base Transactions

To get the base records in a stream before getting new transactions, click on the box to put a check in it. To get just the new transactions, leave the box unchecked. For small tables with relatively few new transactions, it is better to turn this option on. Otherwise, the data for the query will not be displayed. For dynamic tables with high transaction activity, it is better to leave this option turned off. Otherwise, Excel tries to potentially load millions of records every time it starts.

Lossy Subscription

To turn on the this option, click on the box to put a check in it. To leave it turned off, leave the box unchecked. This option is typically used when the network connection between the client and the Sybase Aleri Streaming Platform is slow. If turned on, the client may not get all the transactions if it cannot keep up with the Sybase Aleri Streaming Platform. If turned off, the client receives all the transactions at the expense of potentially slowing down the Sybase Aleri Streaming Platform, especially if the network is slow and the subscription buffer is filled up.

Streams

Displays all the streams available in the server with which the selected **Connection** is associated. This box is automatically populated on choosing a connection.

Columns

When one of the streams is selected, this area displays each column, along with its data type and a check box to indicate whether or not it is a key column. You can choose a different key column for the stream than the specified one.

SQL Statement

If you wish to customize what is retrieved from the Sybase Aleri Streaming Platform, specify an SQL statement in this textbox. The statement cannot include Joins, Group By and Order By clauses as the SQL is being applied to the individual transaction logs for the stream, *not* the data in the stream. See the documentation for **sp_query** for information on the supported SQL syntax. The **SQL Statement** textbox is only enabled when the <<SQL Statement>> entry is selected in the **Streams** listbox.

Parse SQL

Click on this button to have the SQL statement typed in the **SQL Statement** textbox parsed. If the SQL is parsed successfully, the column names and corresponding data types are displayed in the **Columns** listbox. Note that even though by default none of the columns are marked as key fields, the appropriate key columns need to be selected before being able to apply the real-time data query.

Apply

Click on this button to apply the real-time data formulas in the Excel worksheet after configuring a new subscription or modifying an existing subscription. Once the formulas have been applied, the **Start** button is displayed and the query can be started.

Reset

Click on this button to display the properties of the Subscription Query when it was last saved. Use this button if changes have been made to the query that need to be completely reversed.

Delete

Click on this button to delete a previously saved query.

Start

Click on this button to start the query and the data will appear in the Excel worksheet.

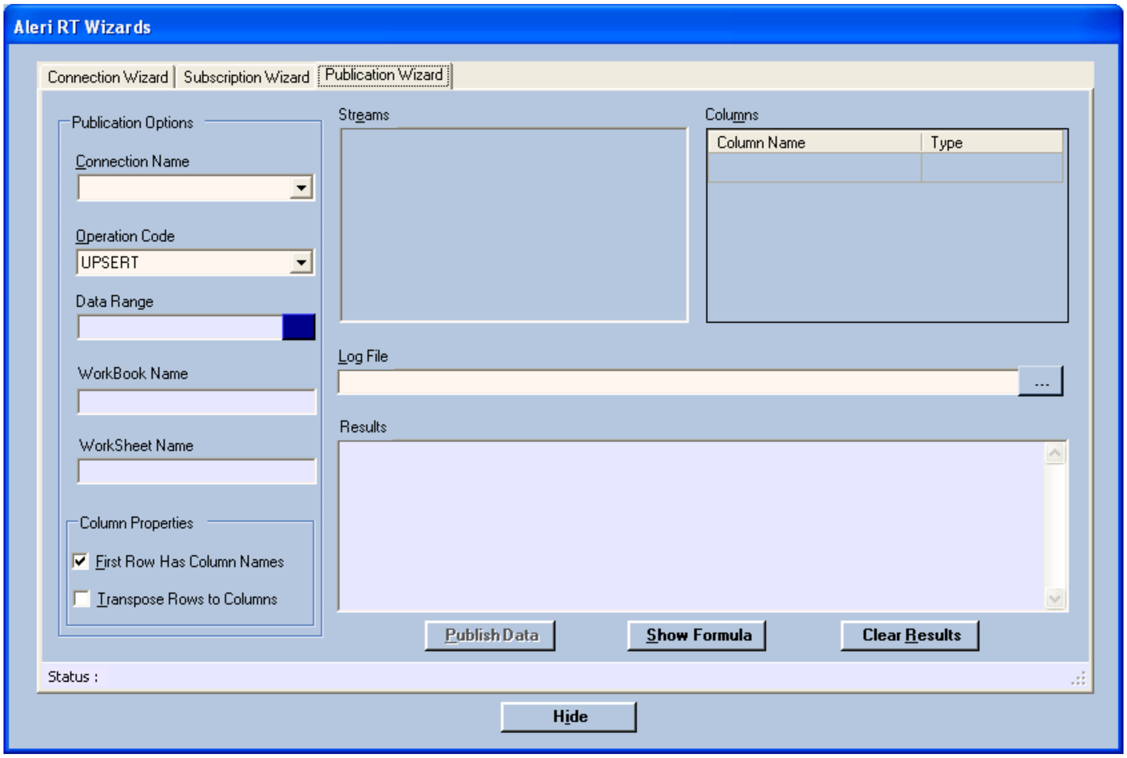
Stop

Click on this button to stop the running query and no more data will appear in the Excel worksheet. However, any displayed data continues to be displayed until the worksheet is closed and reopened or the query starts again.

AleriRT keeps track of the locations of your queries. That means your queries work even if its defined cells are shifted horizontally or vertically.

2.1.3. The Publication Wizard

The Publication Wizard provides a way to manually publish data and graphically construct publication formulas meant for automatic publishing. The following figure shows the Publication Wizard:



Operation Code: UPSERT

☑ First Row Has Column Names

☐ Transpose Rows to Columns

This screen has the following components:

Connection Name

This is the name of the connection to use for publishing. Only active connections are displayed in this box. When a connection is clicked, the streams the connection object is connected to are displayed in the **Streams** list box and the columns and the data types for the stream are displayed in the table named **Columns**.

Operation Code

Select INSERT, UPDATE, DELETE or UPSERT from this list box. If none is selected, the default Operation Code is UPSERT. (If the record exists, update it; if not, insert it.)

Data Range

Specify the range of cells in the Excel worksheet containing the data to publish. This field cannot be edited directly. You must select the cells in the worksheet to publish and then click the blue button next to this field in order to populate it.

Publishing multiple non-contiguous areas simultaneously is not supported. Selecting multiple non-contiguous areas in the Excel worksheet and displaying the address of these selected areas in this field causes an error when attempting to publish the data.

WorkBook Name

This is a read-only field that shows the workbook where the selected range is located.

WorkSheet Name

This is a read-only field that shows the name of the WorkSheet where the selected range is located.

First Row Has Columns

Click on this checkbox to indicate that the first row in the selected range has column names. Leave it unchecked if it does not. When the column names are provided, the data columns can be in any order and only values for the desired fields need to be supplied. The rest of the columns are automatically filled with NULLS. However, if the column names are not provided, AleriRT expects all the columns in the streams to be provided in the exact same order as defined in the Sybase Aleri Streaming Platform.

Transpose Rows To Columns

Click on this checkbox if the data columns for a record are provided vertically in a single column instead of the horizontally across multiple columns, which is the normal way of representing records. Otherwise, leave the box unchecked.

Streams

This list box contains an automatically maintained list of streams in the Sybase Aleri Streaming Platform to which the selected Connection is connected. Select the stream for which a publication should be made.

Columns

This table is automatically maintained and displays the columns and the corresponding data types for the selected stream. This is provided for information only: you cannot select the columns where to publish from this table. But you can copy the names of the columns and paste them in Excel.

Log File

Specify the path and file name of the log file, either entering it directly into the field or browsing and selecting the filename and path by clicking on the button beside this field. This is an optional parameter that specifies the name of the Log File into which any errors that occur during publication are written. Note that the errors are written to this file in addition to being displayed in the **Result** box.

Result

This box is a read-only box that displays the results of the publication.

Publish Data

Once all the data has been provided, click on this button to publish the data to the Sybase Aleri Streaming Platform. The result of the publication is displayed in the **Result** box.

When the data is published to the Sybase Aleri Streaming Platform, it only acknowledges that the data were received (or not). To find out whether the record was rejected for some reason, such as a duplicate insert or bad data, you can either subscribe to the stream or submit an SQL Query.

Show Formula

Click on this button after providing all the information in this screen to graphically create the formula. This provides a convenient way to create a formula for automatic publishing. If there are no errors, the formula is displayed in the result box. You can then copy this formula and place it in the Excel worksheet to start automatically publishing the data. See Section 2.1.4, “Automatic Publishing” for more information.

Clear Results	Click on this button to clear the **Result** field if there are too many entries.

2.1.4. Automatic Publishing

AleriRT provides a mechanism for publishing data automatically whenever a cell is changed. This is accomplished by using the Add-In function called AleriRTP. AleriRTP is a wrapper function around the underlying Excel real-time data mechanism used for publishing data. The syntax for this formula is as follows:

```
=AleriRTP("ConnectionName","StreamName","OperationCode",DataRange,
 [[ColumnRange],[TransposeRows],["LogFile"],[InstanceNo],[NoResults])]
```

where

ConnectionName The name of the connection to use for publishing. The connection must be established before publishing can successfully take place.

StreamName The name of the stream where to publish data.

OperationCode The operation code that needs to be applied for publish. It can be one of "INSERT", "UPDATE", "DELETE" or "UPSERT".

DataRange The address or name of the Excel range containing the data to publish. The DataRange object MUST NOT be enclosed in double quotes.

[ColumnRange] The optional parameter that specifies the Excel Range Address or Range Name containing the stream column names. This parameter MUST NOT be enclosed in quotes.

[TransposeRows] The optional parameter that specifies whether the data record is specified in a column instead of a row. It can be either `True` or `False`. The default value is `False`.

[LogFile] The optional parameter that specifies the name and location of the log file to which any errors are logged. If not provided, no logging is done.

[InstanceNo] This is for internal use. Always leave this value empty.

[NoResults] This is for internal use. Always set this value to `False` or empty.

For example:

```
=AleriRTP("Connection1","Trades","INSERT",A2:E10, A1:E1,False,"C:\logs\log1.log",,
```

The above formula can be placed in any sheet in the workbook. The only thing to keep in mind is to tell Excel when constructing the formula, from which Workbook and Worksheet the Address is referring to either by selecting the appropriate cells in the desired worksheet or using the [Workbook]Worksheet!A1:E5 format.

Once the formula has been placed in Excel, then any change made to any of the cells causes the entire range to be published. But if you want to publish only when certain cells are changed, you have to make this call inside a custom wrapper that encapsulates the business logic dictating when to call this function.

The return value for this function is an array which is formatted as a string using Excel-style location: {{val11,val12},{val21,22}....}. This formula can then be converted into an Excel-style array object. The string will contain one or more array of elements and each sub element contains two subitems. The array string will contain only one element when there are errors in passed in values. Otherwise it will contain one more element than the number of rows to publish.

The first element in the array string is a summary that indicates whether the publish was completely successful or not. If errors are detected when parsing the formula, a one-element array of the form

```
{{"1","Some error message."}}
```

is returned.

If errors are detected when validating the records, or the publish completed successfully, there will be one more array element than the number of rows to publish. For example if there are two rows to publish and both the records have been successfully published, the array string will look like the following example.

```
{{"0",""},{"0",""},{"0",""}}
```

If only one record was published successfully, and another failed for some reason then the return array string will look like this:

```
{{"1","An error message"},{"0",""},{"1","row level error message"}}
```

2.1.5. Saving Subscription Queries

Subscription Queries can be saved permanently simply by saving the Excel Worksheet containing the formula associated with the query. The next time the Excel Worksheet is opened, the query appears in the AleriRT Subscription Wizard and it can be acted upon normally.

2.2. Applying A Query

When the *Apply* button is clicked, the following sequence of events occurs.

1. **AleriRT** first verifies that the supplied Subscription query name hasn't already been used, and then verifies that the provided Start Cell is a valid Excel cell address. If either of the conditions is false, control returns to the user to resolve the problem.

2. **AleriRT** next constructs Excel real-time data formulas based on the specified subscription query and inserts one formula per cell into the active worksheet. Depending on the query, hundreds of formulas may potentially be inserted. **AleriRT** uses the following logic to insert formulas:

   - Formulas are always inserted as a grid, starting at the specified Start Cell location. Each selected column appears in separate but contiguous columns in the Excel worksheet. The value of `Max Rows` controls the number of rows to which the filter is applied.

   - Soon after the first formula is inserted into the active worksheet, Excel recognizes the real-time data formula and makes a call to the AleriRT server that passes the query information for the first filter. The real-time server looks at the information passed, recognizes it as a new query, and spawns a query object. The real-time data server also stores the passed-in information for future use.

- This process is repeated for every formula of the query, with the exception that the real-time server recognizes that the formula is part of the previously seen query. Therefore, it does not create a new query object. Rather, it stores the information so that it can return the data corresponding to the formula.

When the `Start` button is clicked, the following occurs:

1. AleriRT verifies that the connection to the Sybase Aleri Streaming Platform is active and that the specified query is still valid. If either of these condition is false then it returns.

2. Next, AleriRT spawns a new read thread to read the transaction log data from Sybase Aleri Streaming Platform and stores it in an internal buffer.

3. Every tenth of a second AleriRT reads the transaction logs from the internal buffer, decides whether to insert/update or delete records in a display buffer, based on the user-specified key fields. When there is an insert into the display buffer and the number of records in the buffer is equal to the specified `Max Rows` then the oldest record in the buffer is deleted, the rest of the records are moved up and the record is inserted at end. When a record needs to be updated, an in place update is performed. This insert/update mechanism results in a more stable view of the data in the Excel worksheet and makes it easier to create charts based on the subscribed data.

4. Once the display buffer has been populated, AleriRT notifies Excel that new data is available to be displayed. When it receives a request for the data, it sends the data in a format that Excel can understand and display in the appropriate location in the worksheet.

2.3. Known Issues and Limitations

- When the Max Rows is set to a large value (for example, several thousand rows or more), then the performance degrades. The machine becomes very busy as it attempts to process and complete the request.

- When the Sybase Aleri Streaming Platform is stopped, or the connection is lost due to network failure, the AleriRT screen is not automatically refreshed to reflect the current state of the Query and Connection. However, refreshing the screen by again selecting either the Connection or the Query shows the current state of the selected object.

- You cannot use more than one worksheet containing AleriRT Connection and/or subscription information within the same instance of Excel.

Chapter 3. Server Executables

Name

sp — Streaming Processor, a server for processing relational stream data

Synopsis

`sp` [*`OPTION`*...]

Description

sp starts the Streaming Processor, a server process that accepts relational input from files or sockets, processes the data according to a configuration, and sends the data to publish/subscribe clients. Clients may also connect to an SQL interface (for example, using JDBC or through a native C++ API) and issue ad-hoc SQL queries.

sp is the debug version of the Streaming Processor and **sp-opt** is the optimized version. The optimized version is suitable for production use.

Options

`-a` *`username:password`*

Sets the authorization credentials used for inter-node communication when running a distributed cluster or in high availability(hot spare) mode. If running a distributed cluster, all nodes must share these credentials; when running as a high availability system, both the primary and secondary nodes must share these credentials.

`-B` *`path`*

Set the name of the file to which all rejected records will be written.

`-c` *`port`*

Set the Command and Control port. If the *port* is 0 or out of the range 1-65535, the program will pick an arbitrary port. The Command and Control interface accepts commands, such as **sp_upload** or **sp_cli**, from clients.

`-C` *`cluster`*

Choose the cluster configuration to use when running as part of a distributed system.

`-d` *`level`*

Set debug level. The valid range is 0-7, with 7 being verbose.

`-DD`

Start the Sybase Aleri Streaming Platform with trace mode on, paused. This is actually the option `-D` with argument "D". Most of the possible arguments for this option are internal and undocumented.

`-DX`

Quick exit. Bypasses the proper destruction of platform objects on exit. This option is useful mostly for running benchmarks with a simple **time** command. This is actually the option `-D` with argument "X". Most of the possible arguments for this option are internal and undocumented.

`-e` *`path`*

Negotiate encrypted links on SQL and Gateway interface sockets. The *`path`* is the full pathname of the directory containing the server private key and certificate. The files must be named `server.key` and `server.crt`.

`-f` *`path`*

Specify the full pathname of the XML configuration file.

`-F` *`path`*

Specify the full pathname of the XML Schema file (the default is

`$PLATFORM_HOME/etc/Platform.xsd`).

`-g` *`port`*
Set the Gateway port. If the *port* is 0 or out of the range 1-65535, the program will pick an arbitrary port. The Gateway interface is responsible for the publish and subscribe mechanism.

`-H` *`pHost:pPort`*
Specify the host name and the command and control port of the primary server (indicating that the server is a secondary server).

`-j` *`path`*
Set the Java CLASSPATH for Java callouts.

`-k` *`path`*
Specify the full pathname of the directory containing the RSA public keys for RSA authentication.

`-K` *`user`*:*`private key file`*
Set the user and the user's RSA private key for RSA authentication in HA mode.

`-l` *`0|1|2|3`*
Control where log messages get sent. Use 0 for no log messages, 1 to send to stderr only, 2 to send to syslog only, and 3 to send to both stderr and syslog.

`-L` *`license key file`*
Specify the full pathname of the license key file (the default is `$PLATFORM_HOME/etc/license.key`).

`-m` *`megabytes`*
Set the maximum virtual memory size (default is 0, meaning unlimited).

`-M` *`module`*
Start the Sybase Aleri Streaming Platform with only those streams and stores that are contained in the specified module.

`-o` *`true|false`*
Optimize; eliminates redundant updates during processing (default false).

`-P` *`precision`*
Set the number of decimal places in output. The default value is 6.

`-q` *`port`*
Set the SQL Listener port. If the *`port`* is 0 or out of the range 1-65535, the program will pick an arbitrary port. The SQL Listener interface provides the means of accepting SQL queries from clients and sending results back to those clients.

`-r` *`true|false`*
Turn access control on or off (the default is false for off).

`-s`
Run as a daemon.

`-S` *`mem|log|stateless`*
Force stores without a "kind" attribute to have a specific kind, either "mem" for memory stores, "log" for log stores or "stateless" for stateless stores.

`-t` *`sec`*
Set the performance timer interval (default 0 for no timer). This produces performance statistics every "sec" seconds, which are sent to the log and which can be displayed graphically in the performance monitoring tool of the authoring environment.

`-T`
Start a module within a clustered model, ignoring the connection topology of the configuration file.

`-V` *`authentication`*
Use the specified type of *`authentication`*. Valid values are none, pam, rsa, and gssapi. The default is none. If `-V pam` is specified, clients connecting to the server will need to use the `-c`

`user:password` option. If `-V rsa` is specified, clients connecting to the server will need to use the `-k rsaDir` and `-c user` options. If `-V gssapi` is specified, clients connecting to the server will need to use the `-G` and `-c user` options.

If you have SonicWall software that creates an additional network interface with a new IP address, running on a Windows server, you may have to disable it in order to run the Sybase Aleri Streaming Platform with Kerberos authentication enabled (`-V gssapi`).

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-v` Print the detailed revision number for the Sybase Aleri Streaming Platform and exit.

Configuration Files

Configuration of the server is done in XML.

Copyright

See Also

sp_cli(1), sp_cnc(1), sp_convert(1), sp_histexport(1), sp_query(1), sp_server(1), sp_sql2xml(1), sp_studio(1), sp_subscribe(1), sp_upload(1), sp_viewer(1)

Bugs

See the documentation for known issues.

Name

sp_clustermgr — cluster manager for the Sybase Aleri Streaming Platform

Synopsis

```
sp_clustermgr
```

Description

The **sp_clustermgr** brings up the interactive cluster management command line interface, which supports simple commands to inquire about the status of a cluster, start a cluster, and stop a cluster.

Commands

set ldPort <port>

Specify the TCP port used to communicate with each of the cluster nodes. Each node must be running an instance of **sp_ld**. for the cluster manager to communicate with it.

set homeDir <home directory>

Specify the home directory for the instance of *sp(-opt)*. This is used only when a cluster is started. The cluster manager passes this to the launch daemon on each node, and those launch daemons issue a chdir(<home directory>) call prior to starting an instance of *sp(-opt)*. The specified directory must have read/execute permissions with respect to the UID of the running launch daemon.

set execDir <path to sp(-opt)>

Specify the fully qualified path to the **sp(-opt)** executable on the remote nodes. The executable must reside in the same location on each of the nodes in a cluster. The recommended practice is to install the software, and then mount this install directory by NFS on all nodes in the cluster.

set logDir <log directory>

Specify the log directory for the instance of *sp(-opt)*. This directory is used on each node to create a log file that captures messages written by **sp(-opt)** to standard output and standard error. A log file containing these messages is created in this directory having the name <cluster>-<module>.log.

The recommended practice is for this directory to also be a shared NFS mount across the different nodes. This places all the cluster log files in the same location, so they may be easily monitored.

set commandLine <command line for sp launch>

Sets the command line used when starting a cluster. The `-M <module>`, `-C <cluster>` and `-c <c&c port>` options need not be specified, as the cluster manager fills these in from its analysis of the configuration file. The `-f <config file>` option must be part of the command line, and should either be a relative path with respect to the <home directory>, or an absolute path. Also The `-a <user>:<passwd>` option must also be specified, as authentication is required when one node connects to another node in the cluster.

show [ldPort|homeDir|execPath|logPath]

Without any arguments specified the show command lists the values for each cluster manager internal variable. If the argument is a specific internal variable, then the value for the variable is displayed.

status <cluster name> <config file>

Scan the configuration file, extract the nodes(including cold spare nodes) for the given named cluster. For each node, contact the **sp_ld** running on that node and query it for all modules running in the named cluster. All status information is displayed as to what modules are running on what nodes, including if the *sp_ld* could not be contacted on a given node.

start <cluster name> <config file>

Scan the configuration file, extract the nodes for the given named cluster. For each node, contact the *sp_ld* running on that node and have it start an instance of **sp(-opt)** for the module bound to that node as specified in the cluster.

stop <cluster name> <config file> <username> <password>

Scan the configuration file, extract the nodes for the given named cluster (including cold spare nodes). For each node, contact the **sp_ld** running on that node and ask if there are any modules running for the given <cluster name>. For each running module on each node, send the command and control file **sendStreamsExit**, which causes each node in the cluster to shut down.

Recommendations

Have an NFS shared directory across all nodes in the cluster of the form `/<prefix-path>/aleri/` with sub-directories `run/ install/` The **run/** directory should contain the configuration file, and be used for the home and log directories. The exec path is `/<prefix-path>/aleri/install/bin/sp[-opt]`.

Copyright

See Also

sp(1)

Name

sp_clustermon — cluster monitor for the Sybase Aleri Streaming Platform

Synopsis

```
sp_clustermon
```

Description

The **sp_clustermon** performs real-time monitoring and maintenance of a cluster. This tool will use all primary and spare nodes to keep the cluster in a fully functional state. It supports detection of failed modules and nodes, with automatic restart of modules.

Options

`-C` *`cluster`*

Specify the cluster (from within the config file) to monitor.

`-c` *`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-f` *`config-file`*

Sets the configuration file defining the cluster.

`-F` *`path`*

Specify an alternate path for the XML schema file (the default is `$PLATFORM_HOME/etc/Platform.xsd`).

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-k` *`privateRsaKeyFile`*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-p` *`sp-ld-port`* Sets the port used for all copies of the launch daemon, **sp_ld**

Copyright

See Also

sp_ld(1), sp_clustermgr(1)

Bugs

See the documentation for known issues.

Name

sp_ld — launch cluster daemon for the Sybase Aleri Streaming Platform

Synopsis

`sp_ld` [*`OPTION`* ...]

Description

The **sp_ld** daemon listens on a bound TCP port and executes commands used in the management of modules within the Sybase Aleri Streaming Platforms clustered environment. It supports a command API to start, stop and report the status of instances of the Sybase Aleri Streaming Platforms. This daemon is used in conjunction with the cluster manager, **sp_clustermgr**, and the cluster monitor, **sp_clustermon**. When run in the foreground (**`see -f`**), all logging goes to stderr, when run as a daemon, logging is done though the syslog facility. Each node of a cluster (including cold spare nodes) should have a single instance of the **sp_ld** application running.

In a production environment, all nodes of a cluster (including cold spare nodes) should start the **sp_ld** daemon from the servers initialization scripts during system boot.

Options

`-p` <*`port`*>	Specify the TCP port that the application binds to.
`-P` <*`root path`*>	Specify the home directory. The daemon issues a chdir(<root path>) when started.
`-f`	Run the program in the foreground and send logging messages to stdout.
`-h`	Print usage.

Copyright

Copyright 2010 Sybase, Inc. All Rights Reserved.

See Also

sp(1)

Name

sp_monitor — read performance data from a running instance of the Sybase Aleri Streaming Platform and print it in XML format on standard output.

Synopsis

`sp_monitor` [*`OPTION`*...]

Description

sp_monitor reads performance data from a running instance of the Sybase Aleri Streaming Platform and displays it on standard output. Monitoring data is only available if the Sybase Aleri Streaming Platform was started with monitoring option **-t**. A set of performance records, one per stream and one per gateway connection, is obtained from the running Sybase Aleri Streaming Platform every n seconds, where n is specified when the Sybase Aleri Streaming Platform is started, (see **sp -t n**). The **Aleri_Clients_Monitor** stream contains basic information about the connected clients but performance-related fields are populated only with the monitoring option.

A record in the following format, is produced for each stream.

```
<Aleri_Streams_Monitor ALERI_OPS="i"  stream="stream1"
    cpu_pct="0.000000" trans_per_sec="0.499451"
    rows_per_sec="1.098791" inc_trans="5" inc_rows="11" queue="0"
    store_rows="2" last_update="2008-08-26 14:17:14" sequence="123"
    posting_to_client="-1"/>
```

where:

ALERI_OPS holds the opcode for the record

stream contains the name of the stream whose stats are reported

cpu_pct is the percentage CPU utilization over the last reporting interval

trans_per_sec is the transaction rate for this interval

rows_per_sec is the rate of row arrivals for this interval

inc_trans is the number of transactions for this interval

inc_rows is the number of new rows for this interval

queue is the number of records in the queue for the stream

store_rows is the number of rows in the table

last_update is the date/time of the last update

sequence is the sequence number of the update (redundant, since the stream name and last_update already provide an unique identification)

posting_to_client is the handle of the gateway client where the stream was trying to post data at the moment, or -1 if none

A record in the following format, is produced for each gateway client.

```
<Aleri_Clients_Monitor ALERI_OPS="u"
    handle="129" user_name="user" ip="127.0.0.1" host="localhost"
    port="12345" login_time="2008-08-26 12:05:01" conn_tag="rdr"
    cpu_pct="0.000000" last_update="2008-08-26 14:17:14"
    subscribed="1" sub_trans_per_sec="0.499451"
    sub_rows_per_sec="1.098791" sub_inc_trans="5"
    sub_inc_rows="11" sub_total_trans="502" sub_total_rows="1018"
    sub_dropped_rows="0" sub_accum_size="0" sub_accum_ops="-1"
    sub_queue="0" sub_queue_fill_pct="0.000000" sub_work_queue="0"
    pub_trans_per_sec="0.000000" pub_rows_per_sec="0.000000"
    pub_inc_trans="0" pub_inc_rows="0" pub_total_trans="0"
    pub_total_rows="0" pub_stream_id="-1"
    />
```

where:

ALERI_OPS holds the opcode for the record

handle contains the handle of this gateway client

user_name contains the user name of this client

ip contains the address from which this client is connected

host contains the host name from which this client is connected, if resolvable

port contains the port from which this client is connected

login_time contains the timestamp when this client logged in

conn_tag contains the connection tag, if any

cpu_pct is the percentage CPU utilization over the last reporting interval by this client's gateway thread

last_update is the date/time of the last update

subscribed is (1) if this client has subscribed or (0) if not

sub_trans_per_sec is the subscription transaction rate for this interval; the envelopes and service messages are also counted equal to transactions

sub_rows_per_sec is the subscription row rate for this interval

sub_inc_trans is the number of subscription transactions/envelopes/messages for this interval

sub_inc_rows is the number of subscription rows for this interval

sub_total_trans is the total number of subscription transactions/envelopes/messages sent

sub_total_rows is the total number of subscription rows sent

sub_dropped_rows is the number of subscription rows dropped due to the client not keeping up

sub_accum_size — for the pulsed subscriptions, the current number of rows collected in the accumulator, to be sent in the next pulse

sub_accum_ops — reserved for the future with the purpose: for the pulsed subscriptions, the number of operations applied to the accumulator since the last pulse (it may differ from the accumulator size, if multiple operations become collapsed in the accumulator); currently is a placeholder with value of -1

sub_queue — is the number of records in the "proper queue" for this client (the total amount of data buffered consists of sub_accum_size, sub_queue and sub_work_queue)

sub_queue_fill_pct — contains the size of sub_queue in percent relative to its limit

sub_work_queue — is the number of records being transferred from the queue to the socket buffer

pub_trans_per_sec — is the publish transaction rate for this interval; the envelopes and service messages are also counted equal to transactions

pub_rows_per_sec — is the publish row rate for this interval

pub_inc_trans — is the number of publish transactions/envelopes/messages for this interval

pub_inc_rows — is the number of publish rows for this interval

pub_total_trans — is the total number of publish transactions/envelopes/messages received

pub_total_rows — is the total number of publish rows received

pub_stream_id — is -1 if the publisher could not write to the stream to which it is currently trying to publish, otherwise it is the numeric id of that stream

Required Arguments

`-p` *`[host:]port`* — Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

Options

`-c` *`user[:password]`* — Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-`

V rsa or -V gssapi option, specify -c user.

-G

Use Kerberos authentication. This option is required when the Streaming Processor was started with the -V gssapi option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

-k *privateRsaKeyFile*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the -V rsa option. With this option enabled, the user name must be specified with the -c option, but the password is not required. In addition, the Streaming Processor must have been started with the -k option specifying the directory in which to store the RSA keys.

Examples

To monitor an instance of the Sybase Aleri Streaming Platform running on the host "amazon.aleri.com" with a Command and Control port of 31415, use the following:

```
sp_monitor -p amazon.aleri.com:31415 -c user:pass
```

Copyright

See Also

sp(1)

Bugs

See the documentation for known issues.

Name

sp_playback — A command-line tool that loads data into the Sybase Aleri Streaming Platform from a variety of sources at the specified rate.

Synopsis

`sp_playback` [*`OPTION`*...]

Description

sp_playback is a command line tool that reads data from a variety of formats and loads the data into the Sybase Aleri Streaming Platform. The formats that are currently supported are AleriML, Aleri Delimited, Aleri Binary Format, ODBC Sources and Teradata (using the Parallel Transporter API).

This tool is also capable of playing data at a user specified rate. The rate can be specified either in rows/millisecond or at a rate determined by the values in a timestamp/datetime column in input data. If the latter mechanism is used, the user has the additional capability of specifying a timescale rate, which can be used to speed up or slow down the playback.

This tool is intended to replace the need for **sp_upload** and **sp_convert**, which work only on Sybase file sources.

Options

-a

Use asynchronous publish. In this mode the publication will not wait for the Sybase Aleri Streaming Platform to acknowledge the received data. The default is synchronous publishing.

-c *`user[:password]`*

Authenticate with a *`user`* id and optionally a *`password`*. If neither the *`password`* nor the **-k** option is provided, the user is prompted for the password.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller
2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

-h

Prints a list of possible options on the screen along with a brief explanation for each option.

-i

Turns on the shine through flag. In this mode, if there are any missing columns in an update, they are filled in with the previous values for the updated row. The default behavior is to fill any missing columns with NULLS.

-e

Specifies that communications to the Sybase Aleri Streaming Platform should be encrypted. Note that the Sybase Aleri Streaming Platform must be started in encrypted mode for this to work.

-k *rsaKeyFile*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-p` *[hostname:]port*

Specify the *hostname* and *port*, or just the port of the command and control interface of the Sybase Aleri Streaming Platform. Default hostname is localhost if not provided.

-C connStr

This is a source specific connection string, which specifies the arguments that pertain to the input source being used. The connection string is in the general format *sourceName*:*option1*[:*option2*...]. Note that each token is separated by a colon. If a colon must be supplied in the option, be sure to escape it with a backslash (\). It is a good practice to put the connection string inside single quotes.

The following examples show the connection string formats for the supported sources.

```
odbc:dsnName:{file|sql}:{fileName|query}>
```

where:

odbc

Is the ODBC source string identifier. An ODBC 3.0 compliant driver for the required source must be installed and configured on the machine running this utility to use this source option.

dsnName

Is the ODBC data source name.

file|sql

This specifies whether the following argument is a file name or a SQL query. Only one of these two value must be provided.

fileName|query

The *fileName* is the name of the file containing a SQL statement. The query is the SQL query that needs to be executed to retrieve the data. Only one of these two values must be provided.

```
teradata:host:user:password:configFile
```

where:

teradata — is the identifier for a Teradata database source. To use this source, the Teradata Parallel Transporter API version 12.0 must be installed on the machine on which this utility is running.

This source is currently supported on Microsoft Windows 32 bit and Linux X86_64 environments.

To use it under Linux, make sure that the `$PLATFORM_HOME/lib` path is part of the `PATH` variable. Under Windows, make sure the `%PLATFORM_HOME%\lib` folder is in the `LD_LIBRARY_PATH`.

`host` — is the hostname or IP address of the host machine running the Teradata database server.

user — is the user name to use to connect to the Teradata database.

`password` — is the password corresponding to the user name to connect to the Teradata database.

`configFile` — is the configuration file that specifies the SQL query to use, the table structure corresponding to the data returned by the SQL query (using the TPTAPI datatypes) and any TPTAPI related configuration values. See the section called “Defining a Teradata Configuration File” for more information on creating a configuration file.

```
alerixml:inputFile
```

where:

alerixml — is an identifier for an AleriML file source.

`inputFile` — is the full path and name of the file containing data in AleriML format.

```
aleridlm:inputFile[:delimiter]
```

where:

aleridlm — is an identifier for an Aleri Delimited file source.

inputFile is the full path and name of the file containing data in Aleri Delimited format.

delimiter This optional parameter is a single character field delimiter; the default is a comma.

```
alerixml:inputFile
```

where:

alerixml is an identifier for an AleriML file source.

inputFile is the full path and name of the file containing data in Aleri Delimited format.

```
binary:inputFile
```

where:

binary is an identifier for an Aleri Binary file source. The advantage to this format is that the loads will be faster because there is no conversion required because the data is already in a format that the Sybase Aleri Streaming Platform can absorb. The disadvantage to this format is that it is machine architecture specific.

One can use **sp_convert** to convert data from either AleriML or Aleri Delimited formats into the Aleri Binary format.

inputFile is the full path and name of the file containing data in Aleri Binary format.

```
recorder:inputFile
```

where:

recorder is an identifier for a file generated by the Recorder. The recorder can be started via the Aleri Studio or the through the recorder examples in the `$PLATFORM_HOME`/client/pubsub/ folder.

inputFile is the full path and name of the file containing data generated by the recorder.

-R *playRate* This specifies how fast/slow the data must be played back. If this parameter is not supplied, it means play as fast as possible. The following examples show the two ways that the playback rate can be supplied.

```
records:milliseconds
```

where:

records is the number of records to publish in the given number of milliseconds. This value can be 0 only if the *millisecond* component is also 0. A value of 0 indicates play as fast as possible.

milliseconds is the number of milliseconds to playback the given number of records such that *records/milliseconds* gives the number of records to playback in a millisecond.

This property is not supported for source of type Recorder.

```
columnName
```

is the column name in the target stream that controls how fast the record is played back. The column name is case sensitive and a error is reported if the provided column name does not exist in the target Stream.

The columnName property is ignored for source of type Recorder and is currently not supported for Binary file sources.

-T *timeScaleRate* specifies a multiplication factor for the delta between the times for two consecutive records. This is used in conjunction with the playback rate, when the playback rate is controlled by a column in the source. It takes an integer value between -N to +M. A positive value greater than +1 speeds up the time and negative less than -1 slows down the rate. A value of +1 or -1 plays back at the rate specified in the column and a value of 0 specifies that the column values be ignored. Default value is 1.

-r *interval* specifies the minimum number of seconds to wait between each reporting of the publish statistics. The default is 5 seconds. A value of 0 indicates no reporting. Note that the time interval check is triggered when there is a record to be published. This means that even when the reporting interval has been exceeded the statistics will not be reported if there are no records to trigger the check.

-B *bufferSize* is the value is used to specify the internal read and write buffer sizes. Default is 32K, which is the maximum. Note that smaller buffer sizes will use less memory but may run slower under some situations.

-t *size* specifies that transaction blocks must be used to publish data to the Sybase Aleri Streaming Platform, with each block *size* large. The Sybase Aleri Streaming Platform will process data faster if transaction blocks are used to deliver the data to the Sybase Aleri Streaming Platform. The performance boost is achieved in two ways. The first is that the network is used more efficiently because a larger number of records are packed into a single network packet. The second way that the Sybase Aleri Streaming Platform achieves the performance boost is by treating the records as a single block, which fails or passes in its entirety. Treating the records this results in less processing overhead. Depending on the nature of the application, this option may not be suitable.

-w *`size`*

This specifies that envelopes must be used to publish data to the Sybase Aleri Streaming Platform, with each envelope of size *`size`*. If neither the -t nor -w option is specified, the default value is -w64. The value of *`size`* must be between 1 and 1024. Using this option ensures network efficiency in delivering data to the Sybase Aleri Streaming Platform by modifying it to treat the records as individual records.

-H [*`hostname`*:]*`port`*

specifies the *`hostname`* and *`port`*, or just the port of the command and control interface of the hot spare instance of the Sybase Aleri Streaming Platform. If no hostname is provided, the default is localhost.

-s *`streamName`*

specifies the target stream name for this utility. This option is required for the ODBC and Teradata sources. All other sources have the target Stream-name/Stream-Id embedded in them.

The *`streamName`* is case sensitive.

-S *`maxStringSize`*

specifies the maximum string size that can be handled. This option is used in ODBC and Teradata sources. The default value is 1024.

This value is global for all strings in the source. Specifying a large value for this option may result in increased memory usage depending on the number of strings in the record and the value specified with the `-B` option.

-m *`dateMask`*

This is the date mask to use for XML and Delimited files. The default is "%Y-%m-%dT%H:%M:%S'. Note that the date mask is common for all the date columns. This option is only meaningful for the AleriML and Aleri Delimited file sources.

Defining a Teradata Configuration File

The 'teradata' source described above uses the Teradata Parallel Transporter API to load data from Teradata into the Sybase Aleri Streaming Platform in very efficient manner. This source, unlike the other source formats, requires that an XML configuration file be specified that contains the following information:

SQL Statement

The SQL statement that must be executed on the Teradata database to fetch the data that needs to be loaded into the Sybase Aleri Streaming Platform. The SQL statement is a required element and is enclosed between the <SQL></SQL> tags. A sample SQL statement element is provided below.

```
<SQL>
     Select Id, cast(TradeTime as CHAR(23)), Symbol, Price, Shares, "Corr" from
</SQL>
```

There are a couple of items to note in the above example. The first is that the column names used in this query do not matter as they are not used by the playback utility.

The second is that if the target column in the Sybase Aleri Streaming Platform is a timestamp column then the corresponding datetime column in teradata must be cast to a (CHAR(23)). If the target column is of type date then the datetime column must be cast to CHAR(19).

Schema Definition

The schema definition element is a required element that defines the schema of the resulting data when the SQL statement is executed on the Teradata database. A sample schema element is shown below.

```
<Schema>
    <Field name="Id" type="TD_INTEGER" length="4" />
    <Field name="TradeTime" type="TD_CHAR" length="23"/>
    <Field name="Symbol" type="TD_VARCHAR" length="10"/>
    <Field name="Price" type="TD_FLOAT" length="8"/>
    <Field name="Shares" type="TD_INTEGER" length="4" />
    <Field name="Corr" type="TD_INTEGER" length="4" />
</Schema>
```

A schema element can contain one or more Field elements. Each Field element describes a column of data being returned by the SQL statement and it has the following components:

name — is name of the column. This name must match a column name in the target stream in the Sybase Aleri Streaming Platform in order to be used by the Playback utility. The case of the column does not matter, but if the column name does match, then the column is ignored by the Playback utility.

type — is the type of the column. The type of the column must be exactly the same type as the corresponding column in the SQL statement. If it is not an error is generated. Also, the type of the column must be compatible with the target column type in the Sybase Aleri Streaming Platform. If not, an error is generated. The **sp_playback** tool currently supports the following data types:

- TD_SMALLINT
- TD_INTEGER
- TD_FLOAT
- TD_DECIMAL
- TD_CHAR
- TD_BYTEINT

- TD_VARCHAR
- TD_BIGINT

For timestamp types, convert to CHAR(19) or CHAR(23) depending on whether the corresponding type of the column in the target stream in the Sybase Aleri Streaming Platform is a date or a timestamp.

length — is the length of each column. For more information on determining the lengths to specify, see the Teradata™ TPTAPI documentation.

Options — The options correspond to the Teradata TPTAPI options. This is an optional element. There is no limit to the number of options that can be specified. A sample Option element is shown below.

```
<Options>
    <Option name="TD_TENACITY_HOURS" value="0"/>
</Options>
```

The options are specified within a tag called 'Options'. An 'Options' element can have one or more 'Option' elements under it. The Option element itself consists of the following components:

name — This is the name of the option as defined in the Teradata™ TPTAPI documentation.

value — This is a suitable value for the option.

The playback utility, for the most part, passes these options as is to the TPTAPI. However, there are a few options that playback sets by default and these options cannot be overridden. A list of these options follows.

- TD_USER_NAME (taken from the connection string)
- TD_USER_PASSWORD (taken from the connection string)
- TD_TPD_ID

Examples

To read data from an ODBC source with a DSN name of 'foo' using a command line SQL statement into a stream called 'foobar' specify the following command line.

```
sp_playback -cuser:pass -p22200 -C'odbc:foo:query:select * from foobar_db' -sfoobar
```

The following command line will read data from a Teradata database running on a server called myserver, using a config file called `foo.xml` and a target stream called bar.

```
sp_playback -cuser:pass -p22200 -C'teradata:myserver:dbuser:dbpass:/tmp/foo.xml' -sbar
```

To read data from an AleriML source, play the data at a rate of 10,000 rows per second and report the progress every 15 seconds, use the following command line.

```
sp_playback -cuser:pass -p22200 -C'alerixml:/tmp/foo.xml' -sbar -R10000:1000 -r15
```

Copyright

See Also

sp(1), sp_convert(1), sp_upload(1)

Bugs

See the documentation for known issues.

Name

sp_query — Send an SQL query to the Sybase Aleri Streaming Platform and print results on the screen.

Synopsis

`sp_query` [*OPTION*...]

Description

sp_query accepts an SQL query on the standard input and forwards it to a running instance of the Sybase Aleri Streaming Platform. It then prints the results of the query on the standard output.

Options

`-c` *`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-d` *`database`*

The database name when connecting to the Sybase Aleri Streaming Platform. The default is "database". The value of this is ignored.

`-e`

Use an encrypted SSL connection to the Sybase Aleri Streaming Platform.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller
2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-k` *`path`*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-m` *`date format`*

Set the format for date values using strptime format. The default is

"%Y-%m-%d %H:%M:%S+00".

`-q` *`[hostname:]port`* Set the hostname and port of the SQL Listener of the target Streaming Processor. The default host is "localhost" and the default port is "22200".

`-P` *`precision`* Set the number of decimal places in output (default 2).

`-t` *`table name`* Set the name of the table for the XML output. The default name is "Result".

SQL Syntax

The Sybase Aleri Streaming Platform accepts a subset of SQL92 `select`, `insert`, `update`, and `delete` statements. Queries using `select` are limited to single streams, with no joins or subqueries, but may use `where`, `group by`, and `order by` clauses. The `insert`, `update`, and `delete` statements are restricted to source streams. These modification statements can be put in sequence with a semicolon, though.

See the *Programming Interfaces Guide* for more information.

Examples

Suppose the Sybase Aleri Streaming Platform is running on the machine "brule" with SQL port 11100. To print the contents of a stream Emp,

echo "select * from Emp" | sp_query -q brule:11100 -c u:p

To delete an entry from the Dept stream, and update the Emp stream accordingly,

echo "delete from Dept where dn='SWP'; update Emp set dn='' where dn='SWP'" | sp_query -q brule:11100 -c u:p

Copyright

See Also

sp(1), sp_cli(1)

Bugs

See the documentation for known issues.

Name

sp_server — start the Streaming Processor

Synopsis

`sp_server` [*`OPTION`* ...]

Description

sp_server is a shell script that starts up the Streaming Processor (**sp(-opt)**) and the encryption proxy for Command and Control (**sslwrap**).

Options

`-a` *`username:password`*
: Sets the authorization credentials used for inter-node communication when running a distributed cluster or in high availability(hot spare) mode. If running a distributed cluster, all nodes must share these credentials; when running as a high availability system, both the primary and secondary nodes must share these credentials.

`-B` *`path`*
: Set the name of the file to which all rejected records will be written.

`-c` *`port`*
: Set the Command and Control port. This is mandatory and cannot be 0.

`-d` *`level`*
: Set debug level. The valid range is 0-7, with 7 being verbose.

`-e` *`path`*
: Specify the directory containing the SSL keys and certificates. The default value is `/etc/keys`.

`-f` *`path`*
: Set the continuous query specification file. This is mandatory.

`-F` *`path`*
: Specify the full pathname of the XML Schema file (the default is `$PLATFORM_HOME/etc/Platform.xsd`).

`-g` *`port`*
: Set the Gateway port. If the *port* is 0 or out of the range 1-65535, the program will pick an arbitrary port. The Gateway interface is responsible for the publish and subscribe mechanism.

`-H` *`pHost:pPort`*
: Indicates that the server is a secondary server and specifies the host name and the command and control port of the primary server.

`-j` *`path`*
: Set the Java CLASSPATH for Java callouts.

`-k` *`path`*
: Specify the full pathname of the directory containing RSA public keys for RSA authentication.

`-K` *`user`*:*`private key file`*
: Set the user and the user's RSA private key for RSA authentication in HA mode.

`-l` *`0|1|2|3`*
: Control where log messages get sent. Use 0 for no log messages, 1 to send to stderr only, 2 to send to syslog only, and 3 to send to both stderr and syslog.

`-L` *`license key file`*
: Set the license key file (default is `$PLATFORM_HOME/`

`etc/license.key`).

`-m` *`megabytes`*
Set the maximum virtual memory size (default is 0, meaning unlimited).

`-o` *`true|false`*
Optimize; eliminates redundant updates during processing (default false).

`-P` *`precision`*
Set the number of decimal places in output. The default value is 6.

`-q` *`port`*
Set the SQL Listener port. This is mandatory and cannot be 0.

`-r` *`true|false`*
Turn on/off access control (default false for off).

`-S` *`mem|log|stateless`*
Force stores without a "kind" attribute to have a specific kind, either "mem" for memory stores, "log" for log stores, or "stateless" for stateless stores.

`-t` *`sec`*
Turn on performance monitoring statistics, which are updated every <delay> seconds. This option enables the performance monitoring from the graphical user interface.

`-V` *`authentication`*
Use the specified type of *`authentication`*. Valid values are none, pam, rsa, and gssapi. The default is none. If `-V pam` is specified, clients connecting to the server will need to use the `-c user:password` option. If `-V rsa` is specified, clients connecting to the server will need to use the `-k rsaDir` and `-c user` options. If `-V gssapi` is specified, clients connecting to the server will need to use the `-G` and `-c user` options.

`-v`
Print the detailed revision number for the Sybase Aleri Streaming Platform and exit.

Configuration Files

Configuration of the server is done in XML.

Copyright

See Also

sp(1)

Bugs

See the documentation for known issues.

Name

sp_upgrade — convert release 2.x data models to release 3.0 data models.

Synopsis

`sp_upgrade` [*`filename`*]

Description

The **sp_upgrade** command converts an existing data model file for release 2.x of the Sybase Aleri Streaming Platform to a release 3.0 data model file. It reads the data model from the specified filename and writes to standard output. It is invoked automatically by the Aleri Studio to do an in-place update when you open a data model.

It handles most conversion issues automatically; but there are several things that must be done manually.

- converting Rules (**sp_upgrade** comments them out in the XML file)
- converting Exprs (these can appear only in Rules, see previous item)
- converting row local storage to eventCaches

If your data models include any of these, you may want to run the **sp_upgrade** and make the necessary manual changes before opening your data model in the Aleri Studio.

Examples

To convert an existing release 2.x data model named `foo.xml` to a release 3.0 data model named `bar.xml`,

```
    sp_upgrade foo.xml > bar.xml
```

Copyright

See Also

sp_convert(1)

Bugs

See the documentation for known issues.

Name

sslwrap — simple TCP service encryption using TLS/SSL

Synopsis

```
sslwrap [-addr arg] [-cert file|-nocert] [-rsa file]
        [-verify arg] [-Verify arg] [-nbio] [-nbio_test]
        [-debug] [-state] [-cipher arg] [-quit]
        [-no_tmp_rsa] [-ssl2|-ssl3] [-bugs] -port arg
        -accept arg


 -addr arg      Address to connect to (default is
                127.0.0.1)
 -port arg      port to connect to
 -accept arg    port to accept on (default is stdin for
                inetd)
 -verify arg    turn on peer certificate verification
 -Verify arg    turn on peer certificate verification,
                must have a cert.
 -cert arg      certificate file to use, PEM format assumed
                (default is server.pem)
 -key arg       RSA file to use, PEM format assumed, in cert
                file if not specified (default is server.pem)
 -nbio          Run with non-blocking IO
 -nbio_test     test with the non-blocking test bio
 -debug         Print more output
 -state         Print the SSL states
 -nocert        Don't use any certificates (Anon-DH)
 -cipher arg    play with 'openssl ciphers' to see what goes
                here
 -quiet         No server output
 -no_tmp_rsa    Do not generate a tmp RSA key
 -ssl2          Just talk SSLv2
 -ssl3          Just talk SSLv3
 -bugs          Turn on SSL bug
```

DESCRIPTION

sslwrap is a simple UNIX service that sits over any simple TCP service such as POP3, IMAP, SMTP, and encrypts all of the data on the connection using TLS/SSL. It uses ssleay to support SSL version 2 and 3. It can run out of inetd. It can also encrypt data for services located on another computer.

It works with the servers you already have, and does not require any modifications to your existing servers.

Get a certificate

You need a certificate for your server. You can make a self-signed certificate with no encryption using these commands from the ssleay FAQ:

```
cd /usr/bin/ssl
/usr/bin/ssl/openssl req -new -x509 -nodes \
   -out /etc/sslwrap/server.pem -keyout \
   /etc/sslwrap/server.pem \
   -days 365
ln -s server.pem `/usr/bin/ssl/openssl -x509 \
   -noout -hash > server.pem`.0
```

There is information on getting a real server certificate from a Certificate Authority (CA) in the ssleay FAQ. Note that Verisign previously would not issue a certificate for a server using ssleay; though this may have changed.

There is a security problem with sslwrap: it requires an unencrypted private key. Since sslwrap runs out of inetd it is not particularly convenient to prompt the server operator for the private key password. I'm only using sslwrap for the link encryption (not server identity verification), so I'm using a self-signed certificate, and I'm not as concerned about the private key being stolen.

You can use self-signed certificates using Netscape Navigator 2.0 or later, or Microsoft Internet Explorer 3.02 or later. The client will need to go through several dialogs to add the certificate, either for the session or until expiration.

You will want to chmod 600 your certificate file so that normal users won't be able to read your unencrypted private key.

Also, when req prompts for you “Common Name (eg, YOUR name) []” enter your host name, not your name, for a server certificate. Netscape 3.0.2 and later allow wildcards (for example, “*.acme.com”) but Microsoft Internet Explorer 4.0 does not. The hostname can be an IP CNAME, but must match whatever you specified to connect to (in the https URL, mail configuration, and other places) or you will get a warning dialog in the client.

Add to /etc/services

According to IANA, the following port numbers have been assigned for SSL:

```
https 443/tcp     # http protocol over TLS/SSL
ssmtp 465/tcp     # smtp protocol over TLS/SSL
nntps 563/tcp     # nttp protocol over TLS/SSL
telnets 992/tcp   # telnet protocol over TLS/SSL
imaps 993/tcp     # imap4 protocol over TLS/SSL
ircs 994/tcp      # irc protocol over TLS/SSL
pop3s 995/tcp     # POP3 protocol over TLS/SSL
ftps-data 989/tcp # ftp protocol, data, over TLS/SSL
ftps 990/tcp      # ftp protocol, control, over TLS/SSL
```

If you do not have the entries above in `/etc/services`, you will probably want to add them.

Running out of inetd

If you want to run sslwrap out of inetd, you will need to edit `inetd.conf` to add all of the services you want to front-end:

```
https stream tcp nowait sslwrap /usr/sbin/tcpd \
   /usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 80
imaps stream tcp nowait sslwrap /usr/sbin/tcpd \
   /usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 143
telnets stream tcp nowait sslwrap /usr/sbin/tcpd \
   /usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 23
pop3s stream tcp nowait sslwrap /usr/sbin/tcpd \
   /usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 110
```

The service (https, imaps, telnets, pop3s, or other) is the service identifier you added to `/etc/services`.

Running as daemon

You can also run sslwrap as a daemon instead of running out of inetd.

```
/usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 80 \
   -accept 443 &
/usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 143 \
   -accept 993 &
/usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 23 \
   -accept 992 &
/usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 110 \
   -accept 995 &
```

You might type in commands like this, or perhaps add it to one of the startup files, like `/etc/inet.d`.

Connection to another machine

You can “ssl-ize” a service running on a computer other than the one you're running sslwrap on. Of course anyone between you and the server you are connecting on can still look at your clear text data, but if you'll be connecting to the SSL server from an even more distant or insecure area, you may still have benefits from doing this.

Remote connection can be done from inetd or as a daemon; add a “-addr” parameter to the command line:

```
imaps stream tcp nowait sslwrap /usr/sbin/tcpd \
   /usr/sbin/sslwrap -cert /etc/sslwrap/server.pem -port 143 \
   -addr 123.45.67.89
```

The address must be specified in dotted decimal notation and \ be an IP address, not a hostname.

NOTES

Note that you cannot front-end ftp data connections with sslwrap nor can you front-end UDP services. While you can front-end telnet using sslwrap, only the “sslonly” variant of telnet is supported. Other SSL telnet implementations such as SSL-MZtelnet also support SSL via telnet option negotiation to the standard telnet port (23) instead of using the special port (992).

While you can front-end your HTTP server with sslwrap, you're better off using one of the Apache with SSL variations of Apache. It is more efficient and won't adversely affect your logs. If you use sslwrap, all connections appear to come from “localhost”. The Stronghold version of Apache also comes with a Thawte certificate. Using a self-signed certificate for electronic commerce is probably not a good idea.

This man page is not part of the sslwrap source package. It was created for the Debian GNU/Linux distribution.

SEE ALSO

sslwrapconfig(1)

AUTHOR

Debian GNU/Linux, Raphael Bossek <bossekr@debian.org> sslwrap is written by Rick Kaseguma <rickk@rickk.com>

More information can be found at the sslwrap web site at http://www.rickk.com/sslwrap/

Chapter 4. Command and Control Executables

Name

sp_cli — Sybase Aleri Streaming Platform Command line utility

Synopsis

-p *[host:]port* [*OPTION*...] [*COMMAND*...]

Description

sp_cli is a command-line utility that is used to control and get information from a running Sybase Aleri Streaming Platform instance. For example, it can be used to stop the Sybase Aleri Streaming Platform, get the list of streams and their definitions, or get the host and port of the Gateway interface.

Required Arguments

`-p` *[host:]port* — Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

Options

`-c` *user[:password]* — Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-e` — Encrypt messages between the server and **sp_cli** via an OpenSSL socket.

`-G` — Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller
2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-h` — Prints out detailed help.

`-i` *file* — Run the commands in the given file.

`-k` *`privateRsaKeyFile`* Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-q` Disable the prompt. Particularly useful when reading commands from a file piped to standard input on Windows®. (which can't tell that it's not getting the input from a terminal and disable the prompt automatically).

`-x` Enable echoing of commands before execution. This can also be changed later with the command **echo on|off**.

Commands

When a command takes arbitrary values as parameters, they should be quoted. The **sp_cli** command provides two kinds of quotes: back quotes (`) and curly braces {}. For example, `` `parameter` `` or `{parameter}`. (These quotes are used to avoid confusion with the normal single and double quotes that are already used in the syntax of the shell and SPLASH expressions.) Each of these quotes has its own limitations.

The strings enclosed in back quotes may not have back quotes in the middle. This is usually not an issue, since there is no use for back quotes in SPLASH and SQL. A more serious limitation is that the **sp_cli** commands cannot be specified on the command line in double quotes. If they were, the UNIX shell would try to interpret the back quotes. To prevent this, in this case the back quotes need to be shielded with back slashes (\`). For example:

```
        sp_cli -c user:password -p 12345 "history \`100\`"
```

The strings enclosed in braces ({}) must have a balanced number of braces inside them, just like in Tcl. For example:

```
{ row.value = '{a}' }
```

If the parameter contains an unbalanced number of braces, the only way around is by using the back quotes instead. For example:

```
` row.value = '{' `
```

The quoting styles can be mixed within the same command. Different parameters of the same command may use different quoting styles. But they can't be mixed within the same parameter. The command descriptions mostly show the back quotes to avoid confusion with "{" as a metacharacter. But the braces quoting can be used instead, and would be more convenient in most of the practical cases.

Unlike shell and Tcl, back quotes and curly braces cannot be used for multi-line values. Yet another quoting style is intended for the large multi-line parameters, such as the inlined configuration files. These parameters start with `<<!`, then from the next line goes the inline text of the parameter, and finally a line containing only `!` on it (without even any whitespace before or after it). This syntax is simil-

ar to the shell's "<<" syntax but the terminating word is fixed to be "!" and can not be changed. The line breaks after the "`<<!`" and before the terminating "`!`" are not part of the parameter, so the usual single-line values can be specified through the syntax as well, if desired. There is also a way to specify multiple inline parameters: separate them with lines containing `!<<!`. For example:

```
load_config_inline_conv {nobackup,nocompat} <<!
 ... text of the model ...
!<<!
 ... text of the conversion model ...
!
```

This syntax is not specific to any particular commands, it can be used with any commands if desired.

Many commands can redirect output to a file using the following syntax.

```
command > `outfile.dat`
command >> `outfile.dat`
command | `filter-program`
```

The ">" operator overwrites an existing file, and ">>" appends to an existing file. The operator "|" pipes the output to a UNIX command pipeline. The file name or the filter program should be quoted, either in back quotes or braces, as usual.

The general commands are:

addrsize

Output the address/pointer size (in bytes) of the connected platform instance (4 bytes for 32 bit addressing, 8 bytes for 64 bit addressing). This value reflects how the connected platform instance was compiled (32 bit vs. 64 bit). For example, a 32 bit platform could be running on a 64 bit host, in which case the **addrsize** command would return the value 4, not 8.

backup

Creates a backup of all the Log Stores. The backup files are created with suffix ".bak". So the file `dynamic.log` would be backed up into the file `dynamic.bak`. The backup files are created as sparse files, with compacted contents.

clear base stream *`stream`*

Clear (delete) the contents of the specified base stream.

clock

Print the current state of the Sybase Aleri Streaming Platform logical clock. For example:

```
current time: 1071014401.018 2003-12-10 00:00:01.018
rate: 6.000 real: 0 stop depth: 0 max sleep: 100
```

The time is printed as both the number of seconds since UNIX epoch and a user-readable value. Rate is the clock rate relative to real time: 10 means "10 times faster", 0.1 means "10 times slower". The real flag shows whether the clock matches the system time of the machine where the Sybase Aleri Streaming Platform runs (1), or if the clock has been changed artificially (0).

Stop depth shows how many times the clock has been stopped recursively, or in other words how many times **start clock** would have to be called to actually resume the flow of time. When the clock is running, the stop depth is 0. Max sleep is the period of time, in real milliseconds, that guarantees that all the sleepers discover the changes in the clock rate or time. The calls that change the clock rate do sleep automatically for that long (with the logical clock stopped) to ensure that their effects have been applied cleanly.

clock [**rate** `*value*`] [**time** [**add**] `*value*`]

Change the current time and/or rate of the logical clock. Rate is specified as a floating-point number, the minimum rate is 0.001. Time may be specified as a floating point number of seconds since the UNIX epoch or in the format year-month-day **T**hour:min:sec, or the same with milliseconds: year-month-day**T**hour:min:sec.NNN. The letter "T" is literal, as in the default Sybase Aleri Streaming Platform time format. If the time value is prepended with **add**, it specifies a change to the current time. In this case it must be a floating point number of seconds. Prints the previous state of the clock as it was before executing the command. See the description in the **clock** command.

clock real

Return the logical clock of the Sybase Aleri Streaming Platform to the real time. In other words, to use the system clock of the machine where the Sybase Aleri Streaming Platform is running. The clock can not be set to real time while it's stopped (since a stopped clock would diverge from real time immediately). Prints the previous state of the clock as it was before executing the command. See the description in the **clock** command.

clock stop on pause [`*0*|*1*`]

Show or change the flag that controls whether the Sybase Aleri Streaming Platform logical clock stops when the Sybase Aleri Streaming Platform is paused in trace mode. Prints the previous state of the clock as it was before executing the command. See the description in the **clock** command.

clear base stream `*stream*`

Clear (delete) the contents of the specified base stream.

datesize

Output the size of a date field (in bytes) of the connected Sybase Aleri Streaming Platform instance (4 bytes for 32 bit date representation, 8 bytes for 64 bit date representation).

echo `*string*`

Print the string to standard output.

echo on|off

Enable or disable the printing of commands to standard output before execution.

endian

Output the endian value ("big" or "little") of the machine on which the connected Sybase Aleri Streaming Platform instance is running.

fd

Display the field delimiter value.

fd *delimiter*

Set a new field delimiter value. The delimiter must NOT be quoted. Any non-space character is taken as the new delimiter.

gateway

Output the host and port number of the Gateway interface.

get_config

Get the current running XML configuration and print it on standard output or to a local file by specifying >*file* after the command.

help

Output the general help message.

help flags

Output help for the set of output control flags.

history *`number` [`stream`]*

Change the maximum history size of a stream. The history is collected only with the trace mode on. Every time the trace mode is turned off the history is discarded. As the history collects, only the last number of input and output transaction pairs are kept, the older ones are discarded. The default limit on the Sybase Aleri Streaming Platform is 100.

history ex *`stream`*

Display the current maximum history size of a stream.

idx *`streamName`*

Output the index for the stream specified.

immediate stop

Stop the Sybase Aleri Streaming Platform immediately, without shutting down any streams.

immediate pause

Force the Sybase Aleri Streaming Platform to think that it's in a paused state. This is a last-resort way to examine it if it is internally deadlocked or otherwise frozen. This command corrupts the Sybase Aleri Streaming Platform state. It should only be used when it is already hopelessly stuck.

kill *`handle`*

Kill the client connection with the specified handle. The list of all open connections can be found in the metadata stream Aleri_Clients.

kill every *`name`*

Kill all the client connections with the specified tag name. The list of all open connections and their tag names can be found in the metadata stream Aleri_Clients. The tag name can be specified with option `-m` of such commands as **sp_subscribe** and **sp_upload**.

load_config *[`option,...`] `remoteFile`*

Load the new configuration from this file on the server. If the file contains errors, the error messages will be printed in the log file, the error code returned, and the Sybase Aleri Streaming Platform will be left unchanged.

The options affect the way the changes are applied. The option string consists of options separated by commas (no spaces), for example: *regen,nobackup*. Options to be enabled are listed by names (for example, *regen*), options to be disabled are prefixed with "no" (for example, *noregen*). Some options may have arguments specified after "=" (*optname=value*). When an option takes multiple arguments, the arguments are also separated from each other by equal signs (*optname=arg1=arg2=arg3*). The following options are currently supported:

- **[no]regen**

  Perform/skip the regeneration of the modified streams. The default is **regen**. If the regeneration is skipped, only the processing of the new data will change, and the old data will be left unchanged and thus inconsistent. The allowed modifications with option "noregen" are further limited. The option "noregen" is potentially very dangerous and must never be used in production. It's intended only for very quick and dirty experimentation with models in development. Since the base streams can not be regenerated, this option does not affect them. The regeneration may take a long time to complete. Options **noregen** and **nocompat** are mutually exclusive.

- **[no]backup**

  Perform/skip a backup before changing anything. The default is **backup**. If the backup is skipped and a crash occurs in the middle of the dynamic modification, the data in the Log Stores may be left in an unrecoverable state.

- **[no]compat**

  Limit/don't limit the changes to the existing streams only to the compatible ones, those that don't require the stream to be deleted and then re-created in the new form, with new stream handle. The option to allow incompatible changes is `nocompat` (not `incompat`). Think of it as an abbreviation of “no compatibility check”. The `noregen` and `nocompat` options are mutually exclusive. The default is **compat**.

- **[no]base**

  Allow/disallow the changes on the base streams. Since the data in the base streams can't be regenerated, their contents may become inconsistent with the new configuration. The option "nobase" serves to prevent accidental changes that could cause such issues. The meaning of this option depends on the "compat" option: with "compat,nobase" no changes are allowed on the base streams at all; with "nocompat,nobase" only compatible changes are allowed on the base streams. The default is **nobase**.

- **[no]verbose**

  Has effect on the error messages in the "nocompat" mode: when an incompatible change is found, print the messages about it, even if the change is acceptable and not an error. This can be confusing, since these messages look like errors, but also helpful in diagnostics of dependent changes. The default is **noverbose**.

- **conv=** *remoteFile*

  Requires a conversion performed on the base stream data, and specifies that the data conversion model is in the *remoteFile* on the server. The conversion model is a special temporary model that reads the data from the base streams of the original model, processes it to match the row definitions of the base streams of the modified model, and then places it into the base streams of the modified model (after modification but before regeneration). The typical case for using the conversion model is when the row definition of some base stream changes in an incompatible way. See the more detailed description in the *Administrat-*

or's Guide. The default is empty.

The option list (if present) and the file name must be quoted. The details of allowed configuration changes are described in the *Administrator's Guide*.

load_config_inline *`[option,...]` `XML_model_text`*

Similar to **load_config**, only the new configuration is specified in-line as the last argument of the command, instead of a file on the server. Typically the "<<!" syntax would be used instead of back quotes to specify the argument with the text of the model:

```
load_config_inline  [`option,...`] <<!
 XML text of the model
!
```

The command line must end with "<<!", the following lines contain the XML configuration, and the last terminating line contains only the "!" in it. This syntax is similar to the shell's "<<" syntax but the terminating word is fixed to be "!" and can not be changed.

load_config_inline_conv *[`option,...`] `XML_model` `XML_conversion`*

Similar to **load_config_inline**, but the conversion configuration is also specified in-line instead of a file on the server. Typically the "<<!" syntax would be used instead of back quotes to specify the argument with the text of the model:

```
load_config_inline  [`option,...`] <<!
 XML text of the model
!<<!
 XML text of the conversion model
!
```

The command line must end with "<<!", the following lines contain the XML configuration, then the separator line containing "!<<!" (without any spaces in it), then the lines containing the conversion XML configuration and finally the last terminating line consists of "!". The option **conv** may not be used with this command, the conversion is always implied.

lock timeout *[`seconds`]*

Show or change the value of exclusive lock timeout of the Sybase Aleri Streaming Platform. Most of the commands (except those that wait for certain events) are serialized using the lock, with one command executing at a time. Normally this should happen fast. The timeout prevents the following commands from hanging forever waiting for this lock if something goes wrong. The default timeout is 60 seconds.

loglevel *`level`*

Set the logging level of the Sybase Aleri Streaming Platform.

putd *`delimited SP record`*

Put a single delimited record (within back quotes or braces) to the Gateway I/O process. Refer to "PUT COMMAND NOTES" below.

putx *`XML SP record`*

Put a single XML formatted record (within back quotes or braces) to the Gateway interface. Refer to "PUT COMMAND NOTES" below.

quiesced

Output the quiesced state (1 for true or 0 for false). The state is 1 if there are no publisher connections and all the input data has fully propagated through the model.

quit

Exit out of the sp_cli utility.

refresh_calendars

Refresh calendar data from files; this does nothing if the Sybase Aleri Streaming Platform has not loaded any calendars via its calendar functions.

save_config ` *remoteFile*`

Save the current running XML configuration to this file on the server (file must not exist yet). The file name must be quoted.

setparam ` *param*` `*value*`

Set the specified model parameter to this value.

settings

Output the field separator, flag values, and so forth.

snapshot `*streamName*`

Output the current content of the stream in tabular form, using the output control flag settings (refer to "OUTPUT CONTROL FLAGS" below).

start connector initial

Start all the connectors as specified in the Sybase Aleri Streaming Platform <StartUp> configuration element, just like they get started on the Sybase Aleri Streaming Platform start-up. The connectors that were already running will be left running, the connectors that haven't been running will get restarted. This command waits for all the started connectors to complete their initial loading.

start clock

Resume the logical clock of the platform. Prints the previous state of the clock as it was before executing the command. See the description in the **clock** command.

start connector
`*connector-or-group*`

Start a named connector, or all the connectors from a named group from the <StartUp> element. For the connectors that are already running, this command has no effect. This command does not wait for the full start of the connectors, it returns immediately. Use the command **wait connector initial** to wait for completion of initial loading.

stop

Issue the "exit streams" command to the Sybase Aleri Streaming Platform Command and Control interface, causing the Sybase Aleri Streaming Platform engine to exit.

stop clock

Stop the logical clock of the Sybase Aleri Streaming Platform. The records will still be processed but the notion of time won't change and the timer events won't happen. While the clock is stopped, the time and rate may be changed but the clock may not be switched to real time. Stopping may be called multiple times, then resume must be called the same number of times to have the time flow resumed. Be careful with it, since the Sybase Aleri Streaming Platform may also stop and resume the clock internally, don't resume what you didn't stop. Prints the previous state of the clock as it was before executing the command. See the description in the **clock** command.

stop connector
`*connector-or-group*`

Stop a named connector, or all the connectors from a named group from the <StartUp> element. For the connectors that are already not running, this command has no effect. The output connectors are allowed to complete the processing of their output queue before they get stopped (however as of this command no more new events will be added to this queue). This command does not wait for the connectors to be stopped, it returns immediately. Use the command **wait connector** to wait for completion of the connectors.

stop connector immediate *`connector-or-group`*

Stop a named connector, or all the connectors from a named group from the <StartUp> element. Similar to **stop connector** but has a slightly different effect on the output connectors. Their output queue gets discarded and the connectors are requested to stop immediately. For the input connectors **stop connector** and **stop connector immediate** are equivalent, since these connectors have no output queue. This command does not wait for the connectors to be stopped, it returns immediately. Use the command **wait connector** to wait for completion of the connectors.

stream *`streamName`*

Output the definition of the specified stream, using the "hdr" and "sphdr" output control flags (refer to "OUTPUT CONTROL FLAGS" below).

streams

Output the list of base and derived streams.

throttle *`number` [`stream`]*

Change the input queue throttle value for one or all streams. Any writes to the stream's input queue get blocked when the queue size reaches double the throttle value. The throttle value can not be increased beyond the default value. It can only be reduced. Reducing this value is useful for easier tracing of records during debugging.

throttle ex *`stream`*

Display the current throttle value of a stream.

trace_mode *[on|off]*

Change or get the current state of the trace mode. Without any argument prints the current state, *on* enables the trace mode, *off* disables it. The trace mode is prerequisite for the commands related to single-stepping, breakpoints and examining the debugging information.

wait connector *`connector-or-group`*

Wait for a named connector, or for all the connectors from a named group from the <StartUp> element to terminate. They may be terminated by any reason, either naturally (running out of data in the data source) or as a result of the **stop connector** command.

wait connector initial *`connector-or-group`*

Wait for a named connector, or for all the connectors from a named group, from the <StartUp> element to complete the initial loading. In other words, it waits for the connector state to change to something other than "INITIAL".

wait quiesced

Wait until all the input fully propagates through the model. The new input received after this point will be buffered until the propagation of the previous data completes. Then the Sybase Aleri Streaming Platform resumes normal operation.

wait quiesced gateway

Wait until all the publishing clients disconnect and all the received input fully propagates through the model. This command essentially waits for the condition when the command **quiesced** would return 1. If any new clients connect while this command is waiting for the data to propagate, the data from them will be buffered until the wait completes.

Commands Requiring the Trace Mode

pause

Pause the Sybase Aleri Streaming Platform execution. Returns after the pause begins. All the data examination and single-stepping commands require the Sybase Aleri Streaming Platform to be paused first, explicitly with this command or on a breakpoint or exception on bad data. If the Sybase Aleri Streaming Platform is already paused, this command returns success immediately.

check_pause

Show whether or not the Sybase Aleri Streaming Platform is currently paused.

wait_pause

Wait until the Sybase Aleri Streaming Platform gets paused by a breakpoint or from another instance of **sp_cli**. The wait is not interruptible (other than by killing **sp_cli**).

run

Continue the normal platform execution.

step *[`stream`]*

Do a single step on a paused platform. If a stream name is given as an argument, a single step is done on this stream. Otherwise a stream to be stepped is picked at random among the streams ready to process data. If no streams are ready to process (all of them are waiting for input or output), or with the stream argument if this stream is not ready to process, this command is a no-op and returns success immediately.

step timeout *[`number`]*

Set the timeout in milliseconds for the automatic stepping (as described below). The default timeout is 0.3s. Using a negative or zero value resets the timeout to default.

step trans *`stream` [`limit`]*

Automatically step the stream at least once, and then to just before the end of transaction (the "PUT" location on the stream state diagram). The second argument may be specified to limit the number of steps to be made, to limit the running time in case of very big transactions. The default limit is 10000. If the stream has no input pending, or if it blocks on the output for more than timeout (see above), the stepping will also stop and return an error.

step quiesce stream *`stream` [`limit`]*

Automatically step the stream and all its descendants until all of them are quiesced (their input queues are empty). Note that the first word "stream" above is a literal and the second one represents a parameter, the stream name. The second argument may be specified to limit the number of steps to be made, to limit the running time in case of large amount of data collected in the queues. The default limit is 100000. If the stream is a base stream, and the input on it keeps coming fast enough, the call will return only when the limit of steps is achieved. There is no such danger for a derived stream since the Sybase Aleri Streaming Platform is paused when stepping, any inputs of this stream will be paused too. Though if this stream's input queue was full and any inputs are waiting to deposit their already processed data on it, as the input queue gets processed, these waiting inputs would add their one transaction to the queue. If none of the streams has input pending, or if all of them block on the output for more than timeout (see above), the stepping will also stop and return an error.

step quiesce downstream *`stream` [`limit`]*

Similar to **step quiesce stream**, except the stream itself is not stepped. Only its descendants are stepped. This command is convenient to clear out the descendant streams' input queues. Then when the argument stream will produce its output, the progression of the data through the descendant streams can be traced easily.

step quiesce from base *[`limit`]*

Automatically step all the derived (non-base) streams until their input queues are empty. The argument may be specified to limit the number of steps to be made, to limit the running time in case there is a large amount of data collected in the queues. The default limit is 100000. If none of the streams has input pending, or if all of them block on the output for more than timeout (see above), the stepping will also stop and return an error. This command is useful to clean out the queues of derived streams before processing an interesting record through the base stream. Then the progression of data through the derived streams can be watched easily.

dump *`filePrefix` [`stream`]*

Dump the contents of each stream or one specific stream to a file. Each file is named *filePrefix*dump_*streamName*.xml.

bp add *`stream` `inputStream` [`condition`]*

Add a breakpoint on a stream, before it starts processing an input record from another stream *inputStream*. Optionally, a condition may be specified, and the breakpoint would trigger only when the condition evaluated on the input record is true. The condition is a SPLASH expression. It may refer to two predefined variables: *row* - the current input record, and for the deletes and update blocks in the derived streams *oldrow* - the previous value of the record with this key, that is being updated or deleted. The condition may refer to the fields in the records as usual, "*row.field*". The stream's local and global variables may be used as well.

This command prints the ID of the newly created breakpoint.

bp add *`stream` any*

Add a breakpoint on a stream, before it starts processing an input record from any stream. The condition may not be specified in this case.

This command prints the ID of the newly created breakpoint.

bp add *`stream` out [`condition`]*

Add a breakpoint on a stream, after it has processed an input record and produced some (possibly empty) output. Optionally, a condition may be specified, and the breakpoint would trigger only when the condition evaluated on any of the produced output records is true. The condition is a SPLASH expression. It may refer to one predefined variable: *row* - the current output record. Since one input record may produce multiple output records, the condition is evaluated for each of them in order. If there was no output produced, the condition still evaluates once with *row* set to *NULL*. The condition may refer to the fields in the records as usual, "*row.field*".

This command prints the ID of the newly created breakpoint.

bp del *`id`*

Delete the breakpoint with specified ID (as returned from **bp add** or reported by **bp list**).

bp del all

Delete all the breakpoints.

bp on|off *`id`*

Enable or disable the breakpoint with specified ID.

bp on|off all

Enable or disable all the breakpoints.

bp every *`count` `id`*

Make the breakpoint with specified ID trigger on every Nth occasion. For example, to make the breakpoint with ID 8 trigger on every 100th record, use "**bp every `100` `8`**". Setting the count to 1 makes the breakpoint trigger every time, just like when it was originally created.

bp every *`count` all*

Make all the breakpoints trigger on every Nth occasion.

bp list

List the breakpoints. A convenience alias for "**ex `breakpoints`**". See the details there.

ex *`kind` [`stream` [`object`]]*

Examine data in the Sybase Aleri Streaming Platform. It takes the name of the kind of data, of the stream to which it belongs, and of the particular object. For some kinds of data, the stream and object arguments may not be applicable. The data is printed in XML format, with the element name for most data kinds set to "row". If the data represents a transaction, it's enclosed in a **<trans>** element. If the data represents an update pair, it's enclosed in a **<pair>** element. The exact fields depend on the data being examined.

When examining the input data kinds (input queue, current input transactions and row, input history), the data may be a mix of rows of different types, produced by different streams. In this case, to tell them apart, the name of the XML element is set to the name of the stream that produced it (for base streams it would be the name of the base stream itself).

The following kinds of data are currently supported:

- **`pause`**

  State of the user streams when paused. The fields are:

  - **name**

    Name of the stream.

  - **loc**

    Location where the stream is paused.

  - **onbp**

    If on a triggered breakpoint, the ID of that breakpoint, otherwise 0. If multiple breakpoints were triggered at the same time, will contain the ID of one of them.

  - **throttle**

    The input queue throttle value (see the **throttle** command).

  - **history**

    Maximum size of the kept history.

  - **postSeq**

    Count of transactions ever posted to the input queue.

  - **inSeq**

    Count of transactions ever read from the input queue.

  - **outSeq**

    Count of transactions ever processed to the output (including the empty transactions that get discarded, and the expiry transactions).

  - **stepSeq**

    The count of steps (as defined by the **step** command) made in the trace mode. This includes both single-stepping and running. The changes in this count can be used to find out which streams have changed their state.

- **`pauseAll`**

  Same as **pause**, only includes the metadata streams as well.

- **`breakpoints`**

  Information about all the currently registered breakpoints. The fields are:

  - **id**

    ID of the breakpoint. Never changes throughout the breakpoint's life.

  - **stream**

    Name of the stream on which the breakpoint is defined.

  - **origin**

    Contains the name of the input stream for a breakpoint on a particular input stream, "*" for a breakpoint on input from any stream, and "" (empty) for a breakpoint on output.

  - **expr**

    Conditional expression.

  - **enabledEvery**

  N to trigger the breakpoint on every Nth matching record. See the **bp every** command for details.

  - **leftToTrigger**

    How many matches are currently left for the breakpoint until triggering.

  - **onit**

    1 if the breakpoint is currently triggered, 0 otherwise.

- **`var`** `` *`var-name`*

  Contents of a global variable (one defined in the XML node <Global>). The fields depend on the type of variable. The indexes in the arrays are shown as **Aleri_Index**. The keys in the dictionaries are shown as **Aleri_Key_<field-name>**. The values of records are shown with fields as in the record definition. The simple variables are represented with the **Aleri_Value** field. For structured values, this command may return multiple rows. If a variable is NULL, nothing is returned. For an array, only the elements with non-NULL values are shown. To access streams' local variables, see below the version of this command with the stream name parameter.

- **`listVar`**

  The list of all global variable names. To list streams' local variables, see below the version of this command with the stream name parameter.

  - **name**

    Name of the variable.

  - **type**

    Type of the variable.

- **`store`** *`stream`*

  Contents of a stream's store. The fields are as in the stream's row definition.

- **`outTrans`** *`stream`*

  The current output transaction as it's being built. The fields are as in the stream's row definition.

- **`outRow`** *`stream`*

  Output produced from processing of the previous input row. May contain multiple or no rows. The fields are as in the stream's row definition.

- **`badRows`** *`stream`*

  When the Sybase Aleri Streaming Platform is paused on a bad rows exception, contains these bad rows. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`badRowsReason`** *`stream`*

  For each bad row reported by **badRows** contains an error message explaining why it's bad. The message is in the **reason** field.

- **`outHist`** *`stream`*

  The output transactions from the stream's history. The empty transactions are returned as records with all fields containing **NULL**. There is one-to-one match between the transactions returned by **ex `outHist`** and *ex `inHist`*. The fields are as in the stream's row definition.

- **`lastOutTrans`** *`stream`*

  The newest output transaction in the stream's history. Essentially the same thing as " **ex `outHistLatest` `stream` `0`**", but in case if the history is empty, returns a success with no rows, while outHistLatest returns an error. The fields are as in the stream's row definition.

- **`outHistEarliest`** *`stream` `index`*

  Select an individual output transaction from the stream's history. The index is a number, the index 0 selects the earliest transaction saved in the history, increasing index means later transactions. If there is no transaction with such an index, returns the "No such object" error. The fields are as in the stream's row definition.

- **`outHistLatest`** *`stream` `index`*

  Select an individual output transaction from the stream's history. The index is a number, the index 0 selects the latest transaction saved in the history, increasing index means earlier transactions. If there is no transaction with such an index, returns the "No such object" error. The fields are as in the stream's row definition.

- **`var`** *`stream` `var-name`*

  Contents of a stream's external variable. Only the external variables (those defined in the XML node <Local>) may be examined. This includes the variables of types array, dictionary and eventCache. The variables defined inside the SPLASH blocks exist only when the appropriate methods run, and can't be examined. The fields depend on the type of variable. The indexes in the arrays and eventCaches are shown as **Aleri_Index**. The keys in the dictionaries and eventCaches are shown as **Aleri_Key_<field-name>**. The values of records are shown with fields as in the record definition. The simple variables are represented with the **Aleri_Value** field. For structured values, this command may return multiple rows. If a variable is NULL, nothing is returned. For an array only the elements with non-NULL values are shown. The global variables can not be accessed this way, use the empty stream name to access them.

- **`listVar`** *`stream`*

  The list of all variable names defined on this stream. Applicable only to the streams that are allowed to have the node <Local>. Does not include the global variables.

  - **name**

    Name of the variable.

  - **type**

    Type of the variable.

- **`queue`** *`stream`*

  An input data kind. Contents of the stream's input queue. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`inTrans`** *`stream`*

  An input data kind. The current input transaction that is being processed. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`inRow`** *`stream`*

  An input data kind. The current input row that is being processed. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`queueHead`** *`stream` `index`*

  An input data kind. Select an individual transaction from the stream's input queue. The index is a number, the index 0 selects the transaction at the head of queue, increasing index means next transactions. If there is no transaction with such an index, returns the "No such object" error. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`queueTail`** *`stream` `index`*

  An input data kind. Select an individual transaction from the stream's input queue. The index is a number, the index 0 selects the last transaction at the tail of the queue, increasing index means previous transactions. If there is no transaction with such an index, returns the "No such object" error. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`inHist`** *`stream`*

  An input data kind. The input transaction from the stream's history. There is one-to-one match between the transactions returned by **ex `outHist`** and *ex `inHist`*. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`lastInTrans`** *`stream`*

  An input data kind. The newest input transaction in the stream's history. Essentially the same thing as "**`inHistLatest` `stream` `0`**", but in case if the history is empty, returns a success with no rows, while **`inHistLatest** returns an error. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`inHistEarliest`** *`stream` `index`*

  An input data kind. Select an individual input transaction from the stream's history. The index is a number, the index 0 selects the earliest transaction saved in the history, increasing index means later transactions. If there is no transaction with such an index, returns the "No such object" error. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`inHistLatest`** *`stream` `index`*

  An input data kind. Select an individual input transaction from the stream's history. The index is a number, the index 0 selects the latest transaction saved in the history, increasing index means earlier transactions. If there is no transaction with such an index, returns the "No such object" error. The fields are as in the row definition of the stream that produced the data (or, for a base stream, of the current stream).

- **`hist`** *`stream`*

  This is a mixed representation of history, with both input and output data. Each input transaction is followed by its matching output transaction. In other words, it's an interleaving of the transactions that would be returned by **ex `inHist`** and *ex `outHist`*. The rows in the input transactions are marked with the XML tag of their origin stream name, the rows in the output transaction are marked with the XML tag "row". If you have an input stream named "row", good luck telling them apart.

- **`lastTrans`** *`stream`*

  A mixed input-and-output data kind, see the description of **`hist`** for details. The newest input and output transactions in the stream's history.

- **`histEarliest`** *`stream` `index`*

A mixed input-and-output data kind, see the description of **`hist`** for details. Select an individual transaction pair from the stream's history. The index is a number, the index 0 selects the earliest transaction saved in the history, increasing index means later transactions. If there is no transaction with such an index, returns the "No such object" error.

- **`histLatest`** *`stream` `index`*

  A mixed input-and-output data kind, see the description of **`hist`** for details. Select an individual transaction pair from the stream's history. The index is a number, the index 0 selects the latest transaction saved in the history, increasing index means earlier transactions. If there is no transaction with such an index, returns the "No such object" error.

- **`aggrGroup`** *`aggregationStream`*

  The internal state of an Aggregation Stream, from its group index. Works only for Aggregation Streams that are not optimized to use the additive aggregations (since there is no group index kept for the additive aggregations). The value fields have names from the input stream row definition. The key fields have the same name as in this stream's row definition but with **Aleri_Key_** prepended to them, to avoid conflicts with the data fields. Finally, the index of the record in the aggregation bucket is in the **Aleri_Index** field.

- **`states`** *`patternStream` [`patternNum`]*

  States of the automatons in a PatternStream. Initially a PatternStream has one automaton per defined pattern. As data is received and matched by patterns, a new automaton is cloned for each seen sequence of events that might match the pattern. As complete patterns are found, or the sequences of events are found to not match the patterns, the automatons are destroyed.

  If the optional parameter *patternNum* is present, shows only the automatons for that pattern.

  The fields are:

  - **pnum**

    Number of the pattern that is being parsed by this automaton. The patterns are numbered starting from 0.

  - **instance**

    Instance number of the automaton. As new automatons are cloned, each of them gets an unique instance number. The instance numbers are never repeated (unless the Sybase Aleri Streaming Platform gets restarted). The pair (pnum, instance) allows to identify an automaton during the whole time of its execution.

  - **state**

A number identifying the current state of the automaton. The automatons are linear; they have no loops in their logic, and may visit a state only once. If the state hasn't changed since last examination, this means the automaton hasn't matched any new data. Even though the automatons are linear, the numbers used for states are not sequential.

- **timed**

  If set to 1, this automaton has an expiration timer attached to it. If the timer expires, its pattern match would be considered failed and the automaton would be destroyed. If set to 0, means that this automaton does not expire. The untimed automaton is the very initial automaton of a pattern, used to clone all the others.

- **time_left**

  For a timed automaton, time in seconds left until expiration. For an untimed automaton, always 0. When the Sybase Aleri Streaming Platform is paused, by default the platform logical clock stops. However, if the clock is set to not stop on pause, the timers keep ticking. A value of 0 or negative means that the automaton will expire when the Sybase Aleri Streaming Platform execution resumes.

- **`bindings`** *`patternStream` [`patternNum`]*

  For each automaton in a PatternStream, the pattern variable bindings that have been caused by the data parsed so far.

  If the optional parameter *patternNum* is present, shows only the data for that pattern.

  The fields are:

  - **pnum**

    Number of the pattern that is being parsed by this automaton. The patterns are numbered starting from 0.

  - **instance**

    Instance number of the automaton. As new automatons are cloned, each of them gets an unique instance number. The instance numbers are never repeated (unless the Sybase Aleri Streaming Platform gets restarted). The pair (pnum, instance) allows to identify an automaton during the whole time of its execution.

  - **var**

    Name of the bound variable. Besides the variables as such, the bound events and constants are listed as well. The constants are shown with unique compiler-generated names.

  - **value**

    Value of the bound variable, in string format. The values of bound events are not shown here, they are reported as NULL. Examine the *events* data kind to see the contents of the event rows.

- **`events`** *`patternStream` [`patternNum`]*

  For each automaton in a PatternStream, the events that have been parsed by the automaton so far. This data kind returns a mix of records of different types. The record types are named after the input streams where they came from.

  If the optional parameter *patternNum* is present, shows only the data for that pattern.

  The fields are:

  - **Aleri_Pnum**

    Number of the pattern that is being parsed by this automaton. The patterns are numbered starting from 0.

  - **Aleri_Instance**

    Instance number of the automaton. As new automatons are cloned, each of them gets an unique instance number. The instance numbers are never repeated (unless the Sybase Aleri Streaming Platform gets restarted). The pair (pnum, instance) allows to identify an automaton during the whole time of its execution.

  - **Aleri_Var**

    Name of the event variable.

  - **as in input stream**

    The rest of the fields keep the names as in the row type of the input stream from which they came.

- **`expect`** *`patternStream` [`patternNum`]*

  For each automaton in a PatternStream, the expected records that would advance the automaton to the next state. This data kind returns a mix of records of different types. The record types are named after the input streams from which they came.

  If the optional parameter *patternNum* is present, shows only the data for that pattern.

  The fields are:

  - **Aleri_Pnum**

    Number of the pattern that is being parsed by this automaton.

The patterns are numbered starting from 0.

- **Aleri_Instance**

  Instance number of the automaton. As new automatons are cloned, each of them gets an unique instance number. The instance numbers are never repeated (unless the Sybase Aleri Streaming Platform gets restarted). The pair (pnum, instance) allows to identify an automaton during the whole time of its execution.

- **Aleri_Var**

  Name of the event variable. If preceded by a "!", receiving such a record would cause a pattern mismatch. Otherwise it would advance the automaton to the next state.

- **as in input stream**

  The rest of the fields keep the names as in the row type of the input stream from which they came. Only the fields that are bound to values are shown, the rest are shown as NULL.

exf *`kind` [`stream` [`object`]] `filter`*

Same as **ex**, only specifies a filter SPLASH expression to be evaluated on the Sybase Aleri Streaming Platform side. Only the records for which the filter evaluates to a true (non-zero, non-NULL) value are returned. With filters any transaction and update pair boundaries are lost, each record is returned by itself.

The filter may refer to the predefined variables with names matching the XML tags of the rows when printed. For most of the data kinds it would be the variable *row* which contains the current record to be filtered. For the input data kinds multiple variables get defined, each named after an input stream of the target stream. In this case when evaluating a record, the variable matching the stream of its origin will contain the record and all the other variables will be set to **NULL**. The condition may refer to the fields in the records as usual, for example " *row.field*".

eval *`stream` `block`*

Evaluate a SPLASH statement (not expression!) on a stream, to change the contents of the external variables (those defined in the XML node <Local> or <Global>) of this stream. The variables defined inside the SPLASH blocks of a stream exist only when the appropriate methods run, and can't be modified. Evaluation in context of any computational stream (that is, pretty much any stream type except CopyStream and UnionStream) can be used to modify the global variables.

An important point is that the unit of code evaluated is not an expression but a SPLASH statement, which must be either a simple statement terminated by ";" or a block enclosed in braces "{}". Multiple statements must always be enclosed in a block. Remember that if you use braces to quote the block argument, the outside braces don't count as the block delimiters, they are just sp_cli quotes!

Good examples:

```
`a := 1;`
{a := 1;}
`{ typeof(input) r := [ a=9; |
  b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }`
{{ typeof(input) r := [ a=9; |
  b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }}
```

Bad examples:

```
`a := 1`
{a := 1}
`typeof(input) r := [ a=9; |
  b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r);`
{ typeof(input) r := [ a=9; |
  b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }
```

All the usual SPLASH syntax applies, including defining the temporary variables in the block. All the stream's variables and global variables are visible and may be read or changed in the statement. No streams or stream iterators are visible in the statement.

Remember that the back quotes and curly braces don't allow you to enter multi-line statements. In the previous examples, splitting of the lines represents the wrapping of the line on the terminal. In many cases, the multi-line quoting format would be more convenient:

```
eval {stream} <<!
{ typeof(input) r := [ a=9; |
    b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }
!
```

Operations on eventCaches require special preparation. Normally, the key of the eventCache is determined by the current input record. But in this case there is no input record, so the key is not set and any operations on eventCaches would have no effect. For them to work, the key has to be set manually using the operator **keyCache(ec-variable, record)**. It must be set before performing aggregation operations on the eventCache, as in the example above.

No value is returned from the evaluation.

Output Control Flags

hdr *[on|off]*

Without any argument, display the state of the "include column name header line" flag, otherwise enable or disable the "include column name header line" flag. When enabled, the column name heading is output prior to the tabular data. For snapshots, the field position, name and field type are displayed. If the field is a key field, the field name will be prefixed with an asterisk "*" character.

sphdr *[on|off]*

Without any argument, display the state of the "include header/data prefix" flag, otherwise enable or disable "include header/data prefix" flag. When enabled, the StreamName and OpCode values will prefix each line of the tabular snapshot data. In addition, if the hdr flag is enabled, the header line will include the Sybase Aleri Streaming Platform "StreamName" and "OpCode" field names. The platform header/prefix is used by the "putd" and "putx" sp_cli commands.

txb *[on|off]*

Without any argument, display the state of the "include TRANSACTION BLOCK content" flag, otherwise enable or disable the "include TRANSACTION BLOCK content" flag. When enabled, the output produced by the snapshot command will include all of the messages contained within the Gateway I/O TRANSACTION blocks. If disabled, the tabular snapshot output is generated using only the INSERT messages from the TRANSACTION blocks.

verbose *[on|off]*

Without any argument, display the state of the "verbose" output flag, otherwise enable or disable the "verbose" output flag. When enabled, this flag produces additional output when processing the snapshot commands. In particular, start_sync, end_sync, TRANSACTION block indicators, final snapshot, and record/row count, are produced.

xml *[on|off]*

Without any argument, display the state of the "XML" output flag, otherwise enable or disable the "XML" output flag. When enabled, the output produced by the snapshot command will be in the Aleri XML record format. The XML format is used by the "sp_cli" putx command.

Put Command Notes

The putd and putx commands use the sphdr StreamName and OpCode prefix. In addition, when putting date fields, the date strings are in the format

%Y-%m-%dT%H%M%S

In addition, the `TZ` environment variable is set to "UTC" before the record is uploaded to the Gateway interface.

Usage Notes

In console mode, the user can issue commands from the console, which allows for command-line editing, and command history retrieval. This mode is entered via the following command:

```
sp_cli -p hostName:22000 -c user:password
```

In command-string mode, the user can feed in a double quoted string containing one or more commands. If multiple commands are specified in the double quoted string, each command must be terminated with a semicolon character. When setting the field separator from the command line, enclose the new field separator character within single quotes, and place a space character between the ending single quote and the semicolon.

For example, to produce a comma separated/delimited snapshot of the "BalanceType" table, using a command string:

```
sp_cli -p hostName:22000 -c user:password "fs \`,\` ; sphdr on; snapshot {BalanceT
```

To run the command in input-file mode, specifying a text file (`commands.txt`) containing a list of commands to execute:

```
sp_cli -p hostName:22000 user:password -i commands.txt
```

Copyright

See Also

sp(1), sp_cnc(1), sp_convert(1), sp_subscribe(1), sp_upload(1)

Name

sp_cnc — sample client app for submitting command and control commands to the Sybase Aleri Streaming Platform

Synopsis

`sp_cnc` -C *`command`* -p *`[host:]port`* [*`OPTION`*...]

Description

The **sp_cnc** application connects to the Sybase Aleri Streaming Platform via the Command and Control and Gateway interfaces, and issues simple Command and Control commands to the server. It prints the results on the standard output.

Required Arguments

`-C` *`command`*

The *command* may be one of the following literals: **getGateway**, **getBaseStreams**, **getDerivedStreams**, **getStreamDefinition**, **getAddressSize**, **getDateSize**, **sendStreamsExit**, **augmentSubscriber**, **removeSubscriber**, **isBigEndian**, **isQuiesced**, **isQuiescedNow**, **returnWhenQuiesced**, **setParam**, **getVersion**, **getStreamHandle**.

`-p` *`[host:]port`*

Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

Options

`-c` *`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-e`

Encrypt traffic via openSSL. When this option is not present, no encryption occurs.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-H` *`handle`*
: Specifies the client handle to direct an **augmentSubscriber** or **removeSubscriber** command. This option is required for those commands.

`-h`
: Print detailed help.

`-k` *`privateRsaKeyFile`*
: Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-P` *`name:value`*
: Specifies the new value to be associated with the parameter. Required for the **setParam** command.

`-s` *`stream`*
: Specifies a single stream. Required for the commands **getStreamDefinition**, **augmentSubscriber**, **getStreamHandle** and **removeSubscriber**.

Copyright

Copyright 2010 Sybase, Inc. All Rights Reserved.

See Also

sp(1)

Chapter 5. Publish and Subscribe Executables

Name

sp_archive — Archive data from the Sybase Aleri Streaming Platform to SybaseIQ.

Synopsis

`sp_archive` -f *`configFile`* -p *`[host:]port`* [*`OPTION`*...]

Description

The **sp_archive** command archives data from one or more streams in the Sybase Aleri Streaming Platform to SybaseIQ, either in batch mode or in real-time. It uses the 'LoadTable' feature of SybaseIQ to archive inserts. Updates and deletes are archived via ODBC.

sp_archive subscribes to the streams that need to be archived. On receiving data, it writes the data to an intermediate file either in a delimited format suitable for bulk loading for inserts or as SQL DML statements for updates and deletes. Running **sp_archive** in Data Warehousing mode (-I option) is the only exception to this rule. In this case, the updates are treated as inserts and deletes are ignored.

When **sp_archive** encounters an error, it quits.

sp_archive keeps track of the last transaction that was archived. In the event **sp_archive**, the Streaming Processor, or SybaseIQ stops, on restart **sp_archive** begins archiving data from the point where it left off. In order to achieve this, every stream to be archived must use the Persistent Subscribe pattern, that is, it must have two additional streams associated with it. The first stream is a Log Stream, which contains the transaction logs for the stream to archive; the second stream is a Control Stream. The Control Stream is an input to the Log Stream; its primary purpose is to serve as a gateway into the Log Stream to purge the archived transaction logs. The Aleri Studio can be used to create the Log and Control Streams by using the 'Create Persistent Subscribe Pattern' action.

The Control Stream and the Log Stream have to be persisted in order to guarantee that no transaction is lost if the Sybase Aleri Streaming Platform is brought down. Consequently at least one Log Store has to be created to store these two Streams. The data stream itself must be able to regenerate.

To stop a running **sp_archive** process, enter **Ctrl-C** on the command line.

Required Arguments

`-f` *`configFile`*
Specifies the XML style configuration file that describes the streams that need to be archived, the connection information to SybaseIQ, the SybaseIQ LoadTable options and other information.

`-p` *`[host:]port`*
Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

Options

`-b`
Sets byteswap mode. This means that the server on which the Sybase Aleri Streaming Platform is running has a different byte order than the architecture where **sp_archive** is running. You can use **sp_archive** to connect to machines of differing byte orders, but not differing address size.

`-B` *`batchsize`*
Specifies the commit batch size when archiving data using ODBC

and SQL data manipulation statements. The default is 1000. This does not modify the commit batch size of bulk loaded transactions. To do this, the appropriate option for the 'LoadStatement' must be specified in the *configFile*. The default batch size for bulk loaded transactions is 100,000. It is important to choose a proper setting for this option. If the batch size is too small, it may generate a lot of small intermediate files and make the archiving inefficient. If it is set too large then it may cause problems when committing the data into the database.

`-c` *`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-d` *`delimiter`*

Specifies the delimiter that **sp_archive** uses when it writes intermediate files. By default, it uses HEX 31 as the delimiter.

It is important not to choose a delimiter that is part of the data being archived. Doing so results in the archived data being corrupted or rejected by the bulk load executable.

`-D`

Enables delta mode. If this is not specified, **sp_archive** archives all existing data for the specified streams and exits.

`-e`

Enables encryption via OpenSSL sockets.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-h`

Displays the command line help text.

`-I`

Enables Data Warehousing mode. In this mode, all deletes are ignored and any updates are treated as inserts. This mode is typically used in a data warehousing scenario where data must not be deleted. In order for this option to work, the tables in SybaseIQ must not have any primary keys specified or the primary keys must be auto generated. In this mode the executable guarantees the use of only the bulk load mechanism to load the data.

`-k` *`privateRsaKeyFile`*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-P` *`precision`*

Specifies an integer value between 0 and 6 that defines how many decimal places to print after the decimal point for floating point numbers. The default value is 6.

`-q` *`queueSize`*

It sets the maximum subscription buffer size. The default is 8K. You can set it to a higher value if the data arrives in bursts, and the Sybase Aleri Streaming Platform reports the subscription buffer has filled up. The minimum value is 1001. You should not set the *`Block Size`* parameter in the configuration file to more than 12000.

`-R`

Enables recovery mode. In this mode, **sp_archive** does not connect to the Streaming Processor to get more data. Rather, it archives any data that was written out to disk previously but not archived because **sp_archive** was interrupted. Once the data has been archived, it exits. In this mode, the `-D` option is ignored.

`-T` *`interval`*

Specifies approximately how often **sp_archive** should archive data deltas when archiving in real-time mode. The interval is from the time the last delta archive operation completed.

`-W` *`wait_time`*

Specifies how many milliseconds the Sybase Aleri Streaming Platform should wait for **sp_archive** to consume the base data. The default value is 30000, and the minimum is 1000.

Environment Configuration

For **sp_archive** to work properly, the environment must be set up correctly:

- The ODBC driver for SybaseIQ must either be in the path where it can be found by **sp_archive**, or it must be configured via the ODBC driver manager. An appropriate `.odbc.ini` file (typically in the users home directory) should also be available. This file must contain the connection information for the target database.

- The user account under which **sp_archive** runs must have full permissions on the working directory and the log directory, including the ability to create these directories, if they do not exist.

- The working directory for **sp_archive** must be readable by SybaseIQ. If SybaseIQ is running on a different server than the one on which you are running **sp_archive**, this directory must be shared (for example, on a SAN or NAS) between the two servers. This is required for the 'LoadTable' statement to work properly.

- When archiving data, it is possible that the archive executable may not keep up with the Sybase Aleri Streaming Platform, especially when applying SQL DML to archive data. In this case a large number of intermediate files may be written out to disk. Therefore, the working directory must have sufficient disk space to handle surges in volume.

- The target tables in the destination database must be predefined and must have the same structure as the source stream (that is, the same columns in the same order with the appropriate datatypes). See the section below on datatype conversions between the Sybase Aleri Streaming Platform and SybaseIQ to create the tables. One additional datetime column can be added to the destination table if the archive executable is instructed to populate this column. This column must be the first or the last column. See the next section for more information.

Archive Configuration

The archive is configured using an XML configuration file. A description of the different elements used to configure the **sp_archive** are listed below.

PlatformArchive

```
<?xml version="1.0" encoding="UTF-8"?>
<PlatformArchive xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="file:///home/aleri/etc/sp_archive.xsd">
  archive definition
  .
  .
</PlatformArchive>
```

This is the topmost element. This example assumes that the `sp_archive.xsd` schema file is located in the `/home/aleri/etc` directory. If it is not, replace that path with the actual path to the schema file.

After the header, the objects used in the archive configuration can be defined in any order. The following describes the syntax for each of the objects that can be included.

SybaseIQ

```
<SybaseIQ id="DestinationName" dsn="DsnName">
     <Option...../>
     <Option..../>
        .
        .
</SybaseIQ>
```

where

`DestinationName` is the name of the destination. It may be any name but it must be unique within the configuration file.

`DsnName` is the name of the dsn found in the `odbc.ini` file. This name is case sensitive and must match exactly the way it was entered in the `odbc.ini` file.

This element describes a SybaseIQ target and provides a way to specify the bulk load options specific to the target. More than one element of this kind may be specified, but only one target can be used for a given **sp_archive** instance.

Option

```
<Option name="OptionName" value="OptionValue"/>
```

where

- *OptionName* is the name of the LoadTable statement option. This name is not case sensitive.
- *OptionValue* is the value for the LoadTable statement option.

There may be any number of options specified and a particular option may be repeated more than once. However, only the most recent value will be picked up. Note that **sp_archive** does not attempt to validate the options. But the options will be validated by 'LoadStatement' and any errors will cause 'LoadStatement' and consequently the archive process to exit.

Streams

```
<Streams id="CollectionName">
      <Stream .../>
      <Stream .../>
          .
          .
</Streams>
```

where

CollectionName specifies the name of the collection of streams to be archived. It can be any name as long as it is unique within the configuration file.

This element defines the collection of streams, whose data needs to be archived. There must be at least one <Streams> element defined and in turn this element must contain at least one <Stream> element.

Stream

```
<Stream sourceName="SourceName"
        targetName="TargetName"
        logStreamName="LogStreamName"
        controlStreamName="ControlStreamName"
        [timestampLocation="{first | last}"]
/>
```

where

- *SourceName* is the name of the stream in the Sybase Aleri Streaming Platform. This name is case sensitive (for example, the name must match the name of the stream exactly as specified in the XML model.
- *TargetName* is the name of the Table in the target database. The case sensitivity of this name depends on the type of the destination. For most, if not all the relational databases, this name is not case sensitive.

- *LogStreamName* is a required parameter that specifies the name of the stream in the Sybase Aleri Streaming Platform that contains the Transaction Logs for the stream to archive.

- *ControlStreamName* is a required parameter that specifies the name of the Control Stream, which is an input to the LogStream. This stream is used to control the purging of the transaction logs in the LogStream which has been archived. If the persistent subscribe pattern in the Aleri Studio is used to generate the LogStream and the ControlStream then the default ControlStream has the name of *SourceName_*.

- *TimestampLocation* is an optional parameter that specifies that a timestamp needs to be included in the archive. This timestamp may be the first or the last column in the destination table.

Archive

```
<Archive target="DestinationName"
         archiveStreams="CollectionName"
         [dbtempDir="WindowsDirPath"]
         [tempDir="WorkingDirPath"]
         [timestampName="TimestampColumnName"]
         [logDir="LogDirPath"]
/>
```

where

- *DestinationName* specifies the "id" of the SybaseIQ element, which describes the target database server.

- *CollectionName* specifies the "id" of the <Streams> element, which defines information about the Sybase Aleri Streaming Platform streams to be archived.

- *WindowsDirPath* specifies the location that **sp_archive** will use to manage the intermediate archive data files using Windows conventions. It is used when SybaseIQ is running on a Windows server and the Sybase Aleri Streaming Platform is running on a UNIX® or Linux server because this directory must be shared between SybaseIQ and the Sybase Aleri Streaming Platform.

- *WorkingDirPath* specifies the location that **sp_archive** will use to manage the intermediate archive data files. If it does not exist, this directory is automatically created. If no directory name is provided, the default is `archiveTemp` in the current directory.

- *TimestampColumnName* is the name of the timestamp column in the target table that **sp_archive** will use to store the time the archive executable processed a record. This name is not case sensitive for most RDBMS.

- *LogDirPath* is the path where the 'LoadTable' of the bulk load executable will place its log files. This directory is created if it does not exist. The appropriate option for the 'LoadTable' statement must be provided in the <SybaseIQ> element in order for the 'LoadTable' statement to generate the logs in this directory.

Datatype Conversion

The following table shows a suggested mapping between the Sybase Aleri Streaming Platform datatypes and the corresponding datatypes in SybaseIQ. You may choose to use other compatible datatypes if required. The two exceptions to this rule are the date and timestamp datatypes, where the corresponding SybaseIQ datatype must be set to datetime.

Sybase Aleri Streaming Platform Datatypes	SybaseIQ Datatypes
int32	integer
int64	long
string	varchar(n)
date	datetime
timestamp	datetime
double	float
money	float

Sample Configuration Files

Listed below is a sample archive configuration file:

```
<PlatformArchive
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file:///aleri/etc/sp_archive.xsd" >
  <SybaseIQ id="Warehouse" dsn="ArchiveDsn">
    <Option name="FORMAT" value="BINARY"/>
    <Option name="ESCAPES" value="ON"/>
    <Option name="BLOCK SIZE" value="100000"/>
  </SybaseIQ>


  <Streams id="MyArchive">
    <Stream sourceName="Titles" targetName="Titles" \
        timestampLocation="last" controlStreamName="Titles_truncate"
        logStreamName="Titles_log"/>
    <Stream sourceName="Books" targetName="Books_Archive" \
        controlStreamName="Books_truncate" logStreamName="Books_log"/>
    <Stream sourceName="Sales" targetName="Sales" \
        timestampLocation="last" controlStreamName="Sales_truncate"
        logStreamName="Sales_log"/>
  </Streams>

  <Archive target="Warehouse" archiveStreams="MyArchive" \
    tempDir="/tmp/workdir" timestampName="ArchiveTime" \
    logDir="/tmp/logDir/"/>
</PlatformArchive>
```

Listed below is a sample ODBC DSN entry in the `.odbc.ini` file:

```
[ArchiveDSN]
Description = Sybase ODBC Data Source
UID         = dba
PWD         = sql
Driver      = Adaptive Server Enterprise
ENG         = datawarehouse_server
DBN         = books
LINKS       = tcpip(host=dbServer;port=2638)
```

where

- **UID** is the user id for SybaseIQ
- **PWD** is the password for the above user.
- **Driver** is the name of the driver when using ODBC driver manager. This name must match with one of the driver information sections in the `odbcinst.ini` file. This is not required when using the SybaseIQ ODBC driver directly.
- **ENG** is the name of the SybaseIQ server.
- **DBN** is the name of the database.
- **LINKS** is the network protocol used to connect to the database.

Limitations and Known Issues

- When archiving updates and deletes, **sp_archive** may not be able to keep up with the Sybase Aleri Streaming Platform. This is because individual update and delete SQL statements are applied via ODBC. to archive the data.

Copyright

See Also

sp(1)

Bugs

See the documentation for known issues.

Name

sp_convert — Read XML or delimited records from standard input and produce Sybase Aleri Streaming Platform binary records on standard output.

Synopsis

`sp_convert` -f *`configFile`* -p *`[host:]port`* [*`OPTION`*...]

Description

sp_convert converts XML and delimited records into binary records compatible with the Sybase Aleri Streaming Platform. The metadata describing the streams is obtained either from a connection to a running instance of the Sybase Aleri Streaming Platform (via the Command and Control interface) or via an Sybase Aleri Streaming Platform compatible configuration file.

Required Arguments

`-f` *`configFile`*

Specifies the XML style configuration file that describes the data that need to be converted to Sybase Aleri Streaming Platform format.

`-p` *`[host:]port`*

Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

Options

`-b`

Indicates that the machine architecture on which the server (that is consuming our data) is running has the reverse byte order of the machine architecture on which **sp_convert** is running.

`-c` *`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-d` *`separator`*

Read and convert delimited records from standard input instead of the default XML format.

`-e`

Negotiate encrypted OpenSSL sockets for all communication with the Sybase Aleri Streaming Platform. This option requires that the Sybase Aleri Streaming Platform be started in encrypted mode.

`-f` *`file`*

Read and parse the specified Sybase Aleri Streaming Platform configuration file for the metadata required to perform the record conversion. The `-f` and `-p` options are mutually exclusive.

`-F` *`path`*

Specify the full pathname of the XML Schema file (default is `$PLATFORM_HOME/etc/Platform.xsd`).

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-h`

Print detailed help.

`-k` *`privateRsaKeyFile`*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-m` *`format`*

Specifies the format string for date values (in strptime format - default is "%Y-%m-%dT%H:%M:%S").

`-p` *`[hostname:]port`*

Specify the port or the hostname and port of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. This instance of the Sybase Aleri Streaming Platform is queried for the metadata required to perform the record conversion. The `-f` and `-p` options are mutually exclusive.

Input Formats

The following example shows a record in delimited format.

```
StreamName<sep>Operation<sep>column_1..<sep>column_n
```

All columns must be present in the delimited form and the row must end with a line feed character. Operation is a single character, {i|u|d|s|p} for insert, update, delete, safe delete (delete only if the record exists), and upsert respectively.

The following example shows a record in XML format.

```
<StreamName [ALERI_OPS="i|u|d|s|p"] [ALERI_FLAGS="s"]
  column_name="value" ... column_name="value" />
```

If ALERI_OPS is not present, operation is taken as an upsert. There is no requirement on the number of columns present. Those that are missing are taken to have null values. If ALERI_FLAGS is present, it can only have the value "s" indicating that the SHINE flag should be set for the record.

Examples

To convert all XML records in file `foo.xml` to native binary format, and post them to a running instance of the Sybase Aleri Streaming Platform:

```
cat foo.xml | sp_convert -p 11180 | sp_upload -p 11180
```

To convert all comma-separated records in file `foo.csv` to native binary format, and post them to a running instance of the Sybase Aleri Streaming Platform:

```
cat foo.csv | sp_convert -d "," -p 11180 | sp_upload -p 11180
```

To convert all XML records in file `foo.xml` to native binary format, and post them to a running instance of the Sybase Aleri Streaming Platform on a target machine HOST which has a differing byte order than the machine on which **sp_upload** is running:

```
cat foo.xml | sp_convert -b -p HOST:11180 | sp_upload -b -p HOST:11180
```

Copyright

See Also

sp(1), sp_subscribe(1), sp_upload(1)

Bugs

See the documentation for known issues.

Name

sp_histexport — used to migrate data to Live OLAP

Synopsis

```
sp_histexport [options] [ p1=val p2=val... . . pN=val ]
```

Description

The **sp_histexport** utility is used to migrate data from the Sybase Aleri Streaming Platform to the Live OLAP system. The data to be migrated is described in an XML file that is passed as input to this utility via the `-f` option.

Options

`-c`*`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-e`

Enables encryption via openSSL sockets

`-F`*`sp_histexport.xsd`*

Specifies the XSD Schema file for validating the XML configuration file. You must enter the full pathname of the XSD schema file. The default is `$PLATFORM_HOME/etc/sp_histexport.xsd`.

`-f` *`input file`*

The *input file* contains a metadata description along with instructions on how the Sybase Aleri Streaming Platform source data is to be migrated over to the historical vector store.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller
2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-h`

Prints out help.

`-k`*`privateRsaKeyFile`*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-l`*`logdir`*

Specifies the directory where the **sp_histexport** `recovery.log` file is saved. The `recovery.log` file recovers the target vector store directory if a failure occurs during migration.

NOTE: It is important to specify a directory on a disk that has a low probability of running out of disk space or crashing. Similar to the outpath value, the **sp_histexport** application appends a trailing slash at the end of the value that is stored in the logpath attribute when producing the `recovery.log` file.

`-m` *`create_using_sql`*

The migration process creates the vector target output area using the SQL statements specified within the XML input file specified with the -f option. When using this option, the directory specified via the -o option must be empty.

`-o`*`outpath`*

Specifies the target directory of the historical vector store. You must enter the full pathname of the directory. The directory and its contents are usually produced by running a process on the Sybase Aleri Streaming Platform.

NOTE: When producing vector store file names, the **sp_histexport** application will append a trailing slash onto the end of the value that the outpath attribute stores.

`-p`*`[hostName:]portNumber`*

Command and Control host name and port number. The portNumber is a required command line option. If the hostName is not specified, it defaults to localhost.

`-q` *`sqlHostName:sqlPortNumber`*, `-q` *`sqlPortNumber`*

The Sybase Aleri Streaming Platform SQL database server host name and port number. The sqlPortNumber is a required command line option. If the sqlHostName is not specified, it defaults to localhost.

`-r`*`recovery.log file`*

The -r option is used to start up the utility in recovery mode. When the utility is started in recovery mode, it will attempt to perform recovery using the specified `recovery.log` file and then simply exit.

You must enter the complete path to the `recovery.log` file. See the recovery notes to find out more on how the recovery mechanism works.

p1=val1 p2=val2 .... pN=valN

At the end of the command line argument list, a set of optional SQL query substitution parameter name=value pairs can be specified. These name=value pairs will be used to parameterize SQL query statements set up in the `input.xml` file, from the command line at runtime. The name of a substitution parameter must start with a lower case "p" character, followed immediately by one or more digits, followed immediately by an equal '=' sign, that is then followed immediately by the value. Within the XML input file, the SQL statements will look something like this: select BalanceType from BalanceMovements where f1 = ${p1}. At runtime, all of the ${p1} place holders within the SQL queries will be replaced with the value that is assigned to the p1 parameter passed in on the command line.

Usage Notes

The following is an example of how to start this utility without any substitution parameters.

```
sp_histexport -p myhost:3241 -c user:password -s myhost:5434 -f ./dm.xml \
-F ./sp_histexport.xsd -o "/home/tst152/models/alm/staging/output/1/NEW_21" \
-l "/home/tst152/sp_histexport" > output.dat 2>&1
```

The following is an example of how to start this utility with several substitution parameters.

```
sp_histexport -p myhost:3241 -c user:password -s myhost:5434 -f ./dm.xml \
-F ./sp_histexport.xsd -o "/home/tst152/models/alm/staging/output/1/NEW_21" \
-l "/home/tst152/sp_histexport" p1=100 p2="a string" \
p3="\"with double quotes\"" >output.dat 2>&1
```

Output messages generated by the utility go to stderr. If you want to redirect the output messages to an output file, use the “> output.dat 2>&1” syntax at the end of the command line.

The XML input file contains several key entities that are used to describe the overall data migration process. These XML entities are as follows:

Environment

The Environment entity has an attribute called `online` that is set to either true or false. If set to true, where running V-OLAP services will be notified if the migration completes successfully. If the `online` attribute is set to false, running V-OLAP services will NOT be notified when the migration process finishes successfully.

When using the online migration feature, an OlapServices entity will have to be specified within the XML input file and each MigrateStream entity, must have its `host` attribute set. The OlapServices contains a list of host port pairs corresponding to each OLAP service that has to be notified upon the completion of a successful data migration. See the MigrateStream and OlapServices XML entities defined below for more information.

FieldMapDefinition

A FieldMapDefiition has a set of ordered `targetFieldName` attributes. The order in which these `targetFieldName` attributes are defined corresponds to the order of the columns appearing in the SQL query select statement defined within the MigrateStream entity that is using the FieldMapDefinition.

In the historical vector store, the data for each column is stored in a file, the name of which is the number assigned to the `targetFieldName` attribute.

MigrateStream

A MigrateStream entity specifies what Sybase Aleri Streaming Platform source stream data, is to be appended onto a table in the historical vector store. In the historical vector store, each table is represented by a directory, the name of which is represented by a number. The MigrateStream entity has an `id` attribute which is the name of the stream defined in the Sybase Aleri Streaming Platform engine, along with an `targetTable` attribute that specifies the target table name in the historical vector store.

The MigrateStream entity contains a `query` attribute that is set to a select statement that is executed against the Sybase Aleri Streaming Platform SQL database server in order to extract the data that is to be migrated from the Sybase Aleri Streaming Platform database to the historical vector store. Each MigrateStream entity is associated with a FieldMapDefinition, which is a one-to-one mapping between the projection of fields in the SQL query statement and the corresponding fields (vector column files) in the historical vector store.

For online migration support, the MigrateStream entity contains an attribute called `host` that must be set to a value. The `host` attribute setting, along with the directory path of the table being migrated, will be used to produce a "host/directory" message that is sent to each of the online V-OLAP Services specified in the OlapServices entity when the migration finishes. This information is used by the online V-OLAP Services in order to determine the location of the data that was successfully migrated. The value of the `host` attribute here should match the value of the host attribute in the definition of the Service used to specify the fact data partition in LiveOLAP metadata.

OlapServices

The OlapServices entity contains a list of host and port pairs, each representing a running V-OLAP Service. If the `online` attribute of the Environment entity is set to true, each V-OLAP Service will be notified when the migration successfully completes.

Recovery

The recovery mechanism is used to encapsulate the migration process (which is the set of tables specified in the `input.xml` file, for which data is being copied from the Sybase Aleri Streaming Platform and appended onto the specified vector store), within a single recoverable transaction. Either all of the vector store tables are appended, or none of them are. In case of a data migration failure, this mechanism preserves the target vector store area, leaving it in a consistent state (the way it was just before the migration attempt was made).

The utility uses a WAL (Write-Ahead-Log) mechanism to implement recovery. During the migration process, the header of each vector store file is first written out to the `recovery.log` file. After the vector store header has been successfully written out to the `recovery.log` file, the utility attempts to append the Sybase Aleri Streaming Platform data onto the corresponding vector store file. If all goes well, and all vector store files have been successfully appended to, the utility will truncate the `recovery.log` file. If there is a non-fatal error at any point in the migration attempt, the utility will detect the error and attempt to recover/rollback the vector store files to the state that they were in (prior to the migration attempt) before exiting. It does this using the entries in the `recovery.log` file. If a successful recovery is accomplished, the utility will truncate the `recovery.log` file.

If a fatal error occurs, and the utility aborts in the middle of a migration attempt, the next time the utility is started up, the `recovery.log` file is inspected to see if there are any entries in it. If there are, the utility will first attempt to recover/rollback the vector store files using the entries in the `recovery.log` file. Upon a successful start-up recovery/rollback, the utility will truncate the `recovery.log` file and then proceed with the migration (as dictated by the `input.xml` file).

The user can start the utility in "recovery" mode via the -r option. The -r should be followed by the file name (including the path), of the `recovery.log` file. If the application is started up in this mode then just a recovery attempt will be made and the program will exit. If the recovery attempt is successful, the `recovery.log` file will be truncated.

The location/directory of the `recovery.log` file is specified via the `logpath` attribute of the Environment entity of the `input.xml` file.

XML Input File Example

The following is an example of a sp_histexport `input.xml` file. This file would be specified on the **sp_histexport** command line using the -f option.

The `input.xml` file contains several key entities that the **sp_histexport** utility uses to conduct data migration.

The entities shown in XML format are as follows:

```
1)<Environment id="BcgOlapMigrateEnv"
source="platform"
target="olap" />
```

This entity has the following attributes:

- *Environment id* - Identifies the object.
- *source* - String that identifies where the source data is coming from. Currently, it could be any text because the **sp_histexport** system is not using the specified value.
- *target* - String that specifies a target system name. Currently, it could be any text because the **sp_histexport** system is not using the specified value.

```
2)<FieldMapDefinition id="W_ChannelLocationFieldMap">
<Column targetFieldName="3377" />
<Column targetFieldName="3379" />
<Column targetFieldName="3380" />
<Column targetFieldName="3381" />
</FieldMapDefinition>
```

This entity has the following attributes:

- *FieldMapDefinition id* - Uniquely identifies this object as a FieldMapDefinition.
- *Column targetFieldName* - [1,...,n] is an ordered list of aie field numbers, with each field number enclosed in double quotes. In the example above, there are four entries in the FieldMapDefinition.

  NOTE: Each aie field number corresponds to the name of a vector store file located in a subdirectory (representing an aie table) of the specified *outpath*. Later on, this list of field numbers is mapped onto the columns of the “select” statement in the corresponding MigrateStream entity. Source data is retrieved from the Sybase Aleri Streaming Platform through the select statement and each column of source data is appended onto the appropriate vector store file according to the FieldMapDefinition list.

  In addition, a column can have an optional *aieRefField* attribute setting. The *aieRefField* setting defaults to “false”. This setting is only applicable for aie symbol (character) data fields. Note that the provider of the historical vector store application must indicate which column(s) of their FieldMapDefinition(s) must have the optional “aieRefField=true” setting.

```
3)<MigrateStream id="W_ChannelLocation"
targetTable="298"
method="query"
query="select ChannelId, Location, TimeZone, WindowEnd
from W_ChannelLocation"
fieldmap="W_ChannelLocationFieldMap" />
```

This entity has the following attributes:

- *MigrateStream id* - Uniquely identifies this object.
- *targetTable* - Represents the name of a subdirectory of the vector store where the individual column/field files are stored. This subdirectory represents the corresponding aie table.

  NOTE: The `targetTable` must be enclosed within double quotes. The **sp_histexport** utility produces the subdirectory name by appending a slash “/” and the `targetTable` onto the end of the specified `outpath`.
- *method* - Currently, there is only one type of *method* used to retrieve source data from the Sybase Aleri Streaming Platform. This method is called “query”, and it indicates that the source data is retrieved from the Sybase Aleri Streaming Platform through an SQL query.
- *query* - The actual "select" statement.
- *fieldmap* - The name of the FieldMapDefinition list that maps the SQL select statement columns onto the corresponding aie vector store columns. The **sp_histexport** utility produces the path to an individual vector store file by taking the `outpath`, appending a slash “/” character, the “aieTableNumber”, a slash “/” character, and finally the “aieFieldNum”.

The following is an example of the full `input.xml` file (note the use of special aieRefField attributes in the TransactionsFieldMap):

```
<?xml version="1.0" encoding="UTF-8"?>
<PlatformMigration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<!-- Sample XML configuration file for the
Sybase Aleri Streaming Platform

Data Migration sp_histexport Utility. -->

<Environment id="BcgOlapMigrateEnv"
source="platform"
target="olap" />

<!-- W_ChannelLocation table field map -->
<FieldMapDefinition id="W_ChannelLocationFieldMap">
<Column targetFieldName="3377" />
<Column targetFieldName="3379" />
<Column targetFieldName="3380" />
<Column targetFieldName="3381" />
</FieldMapDefinition>

<MigrateStream id="W_ChannelLocation"
targetTable="298"
method="query"
query="select ChannelId, Location, TimeZone, WindowEnd
```

```
from W_ChannelLocation"
fieldmap="W_ChannelLocationFieldMap" />


<!-- W_CurrencyLocation table field map -->
<FieldMapDefinition id="W_CurrencyLocationFieldMap">
<Column targetFieldName="3378" />
<Column targetFieldName="3382" />
<Column targetFieldName="3383" />
<Column targetFieldName="3384" />
</FieldMapDefinition>

<MigrateStream id="W_CurrencyLocation"
targetTable="299"
method="query"
query="select Currency, Location, TimeZone, WindowEnd from
W_CurrencyLocation"
fieldmap="W_CurrencyLocationFieldMap" />


<!-- Transactions table field map -->
<FieldMapDefinition id="TransactionsFieldMap">
<Column targetFieldName="2888" aieRefField="true" />
<Column targetFieldName="3017" />
<Column targetFieldName="3019" />
<Column targetFieldName="3021" />
<Column targetFieldName="3023" />
<Column targetFieldName="3025" />
<Column targetFieldName="3027" />
<Column targetFieldName="3029" />
<Column targetFieldName="3031" />
<Column targetFieldName="3033" />
<Column targetFieldName="3035" />
<Column targetFieldName="3037" />
<Column targetFieldName="3039" />
<Column targetFieldName="3041" />
<Column targetFieldName="3043" aieRefField="true" />
<Column targetFieldName="3045" aieRefField="true" />
<Column targetFieldName="3047" />
<Column targetFieldName="3049" />
<Column targetFieldName="3051" />
<Column targetFieldName="3053" />
<Column targetFieldName="3055" />
<Column targetFieldName="3057" />
<Column targetFieldName="3059" />
<Column targetFieldName="3061" />
<Column targetFieldName="3063" />
<Column targetFieldName="3065" aieRefField="true" />
<Column targetFieldName="3067" />
<Column targetFieldName="3069" />
<Column targetFieldName="3071" />
<Column targetFieldName="3073" />
<Column targetFieldName="3075" />
<Column targetFieldName="3077" />
<Column targetFieldName="3079" />
<Column targetFieldName="3081" />
<Column targetFieldName="3083" />
<Column targetFieldName="3085" />
<Column targetFieldName="3087" />
<Column targetFieldName="3089" />
<Column targetFieldName="3091" />
<Column targetFieldName="3093" />
<Column targetFieldName="3095" />
<Column targetFieldName="3097" />
```

```
<Column targetFieldName="3099" />
<Column targetFieldName="3101" />
<Column targetFieldName="3103" />
<Column targetFieldName="3105" />
<Column targetFieldName="3107" />
<Column targetFieldName="3109" />
<Column targetFieldName="3111" />
<Column targetFieldName="3113" />
<Column targetFieldName="3115" />
<Column targetFieldName="3117" />
<Column targetFieldName="3119" />
<Column targetFieldName="3121" />
<Column targetFieldName="3123" />
</FieldMapDefinition>

<MigrateStream id="Transactions"
targetTable="263"
method="query"
query="select Id, ActualScheduleTime, AdapterStatus,
AssociatedReference, Beneficiary, BeneficiaryBank, Branch,
ChannelId, Correspondent, Counterparty, CRAccountName,
CRNumber, CRSortCode, Currency, CurrentId,
CustomerReference, DealDate, Dealer, DestinationAccount,
DRAccountName, DRNumber, DRSortCode, EventType, Expired,
ExpiryDateTime, ExternalReference, MatchedAmount, MatchId,
MatchIgnored, MatchType, MaturityEvent, MTType,
OrderingInstitution, OriginalAmount, OriginalScheduleTime,
OriginatingSystem, Originator, OriginatorReference,
PartialMatchAllwd, PayReceive, PreviousChannel,
ProcessedDateTime, Product, RemainingAmount, RemittingBank,
ReRouted, ResponsibleUnit, Reversed, SourceSystemInfo,
Status, StreamMovementType, SystemCounterparty,
TransactionType, ValueDateTime, W_ValueDate from
Transactions where trunc(ValueDateTime) = undate('${p1}')"
fieldmap="TransactionsFieldMap" />

<!-- Events table field map -->
<FieldMapDefinition id="EventsFieldMap">
<Column targetFieldName="2887" />
<Column targetFieldName="2889" />
<Column targetFieldName="2891" />
<Column targetFieldName="2893" />
<Column targetFieldName="2895" />
<Column targetFieldName="2897" />
<Column targetFieldName="2899" />
<Column targetFieldName="2901" />
<Column targetFieldName="2903" />
<Column targetFieldName="2905" />
<Column targetFieldName="2907" />
<Column targetFieldName="2909" />
<Column targetFieldName="2911" />
<Column targetFieldName="2913" />
<Column targetFieldName="2915" />
<Column targetFieldName="2917" />
<Column targetFieldName="2919" />
<Column targetFieldName="2921" />
<Column targetFieldName="2923" />
<Column targetFieldName="2925" />
<Column targetFieldName="2927" />
<Column targetFieldName="2929" />
<Column targetFieldName="2931" />
<Column targetFieldName="2933" />
<Column targetFieldName="2935" />
<Column targetFieldName="2937" />
```

```
<Column targetFieldName="2939" />
<Column targetFieldName="2941" />
<Column targetFieldName="2943" />
<Column targetFieldName="2945" />
<Column targetFieldName="2947" />
<Column targetFieldName="2949" />
<Column targetFieldName="2951" />
<Column targetFieldName="2953" />
<Column targetFieldName="2955" />
<Column targetFieldName="2957" />
<Column targetFieldName="2959" />
<Column targetFieldName="2961" />
<Column targetFieldName="2963" />
<Column targetFieldName="2965" />
<Column targetFieldName="2967" />
<Column targetFieldName="2969" />
<Column targetFieldName="2971" />
<Column targetFieldName="2973" />
<Column targetFieldName="2975" />
<Column targetFieldName="2977" />
<Column targetFieldName="2979" />
<Column targetFieldName="2981" />
<Column targetFieldName="2983" />
<Column targetFieldName="2985" />
<Column targetFieldName="2987" />
<Column targetFieldName="2989" />
<Column targetFieldName="2991" />
<Column targetFieldName="2993" />
<Column targetFieldName="2995" />
<Column targetFieldName="2997" />
<Column targetFieldName="2999" />
<Column targetFieldName="3001" />
<Column targetFieldName="3003" />
<Column targetFieldName="3005" />
<Column targetFieldName="3007" />
<Column targetFieldName="3009" />
<Column targetFieldName="3011" />
<Column targetFieldName="3013" />
<Column targetFieldName="3015" />
</FieldMapDefinition>

<MigrateStream id="Events"
targetTable="262"
method="query"
query="select UniqueId, ActualScheduleTime, AdapterStatus,
AssociatedReference, Beneficiary, BeneficiaryBank, Branch,
CalculatedId, CaptureDateTime, CaptureOperator, ChannelId,
ActualScheduleTime, Correspondent, Counterparty,
CounterpartyAlias, CRAccountName, CRNumber, CRSortCode,
Currency, CurrentId, CustomerReference, DealDate, Dealer,
DestinationAccount, DRAccountName, DRNumber, DRSortCode,
EventDateTime, EventType, Expired, ExternalReference,
FrontOfficeId, InterfaceId, MaturityEvent, MTType,
OrderingInstitution, OrderingInstitutionAlias,
OriginalAmount, OriginalId, OriginalScheduleTime,
OriginatingSystem, Originator, OriginatorAlias,
OriginatorReference, PartialMatchAllwd, PreviousChannel,
PreviousId, ProcessedDateTime, Product, RemittingBank,
ReRouted, ResponsibleUnit, Reversed, Rollable,
RollableFromId, SourceSystem, SourceSystemEventInfo,
SourceSystemInfo, StatusWeight, StreamMovementType,
SystemCounterparty, SystemCounterpartyAlias, Text,
TransactionType, ValueDateTime from Events where
trunc(ValueDateTime) = undate('${p1}')"
```

```
fieldmap="EventsFieldMap" />

</PlatformMigration>
```

XML Input File Example (for Online Migration)

The following `input.xml` file example builds off the previous one, showing those sections that would have to change to cause an online migration to take place. This file would be specified on the **sp_histexport** command line using the -f option.

The `input.xml` file contains several key entities that the **sp_histexport** utility uses to conduct an online data migration.

The entities shown in XML format are as follows:

```
1)<Environment id="OnlineBcgOlapMigrateEnv"
source="platform"
target="olap"
online="true" />
```

The following are the additional Environment entity attribute settings required to run an online migration:

- *online* - When set to true, this indicates that an online migration is to take place.

```
2)<FieldMapDefinition id="W_ChannelLocationFieldMap">
<Column targetFieldName="3377" />
<Column targetFieldName="3379" />
<Column targetFieldName="3380" />
<Column targetFieldName="3381" />
</FieldMapDefinition>
```

For online migrations, the FieldMapDefinition entities remain unchanged (refer to the previous `input.xml` file example):

```
3)<MigrateStream id="W_ChannelLocation"
targetTable="298"
method="query"
host="foobar"
query="select ChannelId, Location, TimeZone, WindowEnd
from W_ChannelLocation"
fieldmap="W_ChannelLocationFieldMap" />
```

The following are the additional MigrateStream attribute settings required to run an online migration:

- *host* - This is used along with the `outpath` specified with the -o option and the `targetTable` attribute to build a "host/directory" message which is sent to the online V-OLAP Services (specified in the OlapServices entity), indicating to the running OLAP services, indicating to them the location of the migrated data.

```
4)<OlapServices id="OlapServicesList">
    <Service host="olapmachine" port="11106" />
  </OlapServices>
```

This entity has the following attributes:

- *Service host*="machine_of_olap_service" *port*="port_number_of_olap_service" Each Service entry contains a tuple consisting of a host and port pair, indicating the location of a running V-OLAP Service. Within the encapsulating OlapServices entity, you can specify many Service entries.

The following is an example of the full `input.xml` file (note the changes that were made for the on-line migration in the Environment and MigrateStream entities, and the new OlapServices entity located at the bottom of the `input.xml` file):

```
<?xml version="1.0" encoding="UTF-8"?>
<PlatformMigration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<!-- Sample XML configuration file for the
Sybase Aleri Streaming Platform

Data Migration sp_histexport Utility. -->

<Environment id="BcgOlapMigrateEnv"
source="platform"
target="olap"
online="true" />

<!-- W_ChannelLocation table field map -->
<FieldMapDefinition id="W_ChannelLocationFieldMap">
<Column targetFieldName="3377" />
<Column targetFieldName="3379" />
<Column targetFieldName="3380" />
<Column targetFieldName="3381" />
</FieldMapDefinition>

<MigrateStream id="W_ChannelLocation"
targetTable="298"
method="query"
host="foobar"
query="select ChannelId, Location, TimeZone, WindowEnd
from W_ChannelLocation"
fieldmap="W_ChannelLocationFieldMap" />


<!-- W_CurrencyLocation table field map -->
<FieldMapDefinition id="W_CurrencyLocationFieldMap">
<Column targetFieldName="3378" />
<Column targetFieldName="3382" />
<Column targetFieldName="3383" />
<Column targetFieldName="3384" />
</FieldMapDefinition>
```

```
<MigrateStream id="W_CurrencyLocation"
targetTable="299"
method="query"
host="foobar"
query="select Currency, Location, TimeZone, WindowEnd from
W_CurrencyLocation"
fieldmap="W_CurrencyLocationFieldMap" />


<!-- Transactions table field map -->
<FieldMapDefinition id="TransactionsFieldMap">
<Column targetFieldName="2888" aieRefField="true" />
<Column targetFieldName="3017" />
<Column targetFieldName="3019" />
<Column targetFieldName="3021" />
<Column targetFieldName="3023" />
<Column targetFieldName="3025" />
<Column targetFieldName="3027" />
<Column targetFieldName="3029" />
<Column targetFieldName="3031" />
<Column targetFieldName="3033" />
<Column targetFieldName="3035" />
<Column targetFieldName="3037" />
<Column targetFieldName="3039" />
<Column targetFieldName="3041" />
<Column targetFieldName="3043" aieRefField="true" />
<Column targetFieldName="3045" aieRefField="true" />
<Column targetFieldName="3047" />
<Column targetFieldName="3049" />
<Column targetFieldName="3051" />
<Column targetFieldName="3053" />
<Column targetFieldName="3055" />
<Column targetFieldName="3057" />
<Column targetFieldName="3059" />
<Column targetFieldName="3061" />
<Column targetFieldName="3063" />
<Column targetFieldName="3065" aieRefField="true" />
<Column targetFieldName="3067" />
<Column targetFieldName="3069" />
<Column targetFieldName="3071" />
<Column targetFieldName="3073" />
<Column targetFieldName="3075" />
<Column targetFieldName="3077" />
<Column targetFieldName="3079" />
<Column targetFieldName="3081" />
<Column targetFieldName="3083" />
<Column targetFieldName="3085" />
<Column targetFieldName="3087" />
<Column targetFieldName="3089" />
<Column targetFieldName="3091" />
<Column targetFieldName="3093" />
<Column targetFieldName="3095" />
<Column targetFieldName="3097" />
<Column targetFieldName="3099" />
<Column targetFieldName="3101" />
<Column targetFieldName="3103" />
<Column targetFieldName="3105" />
<Column targetFieldName="3107" />
<Column targetFieldName="3109" />
<Column targetFieldName="3111" />
<Column targetFieldName="3113" />
<Column targetFieldName="3115" />
<Column targetFieldName="3117" />
<Column targetFieldName="3119" />
```

```
<Column targetFieldName="3121" />
<Column targetFieldName="3123" />
</FieldMapDefinition>

<MigrateStream id="Transactions"
targetTable="263"
method="query"
host="foobar"
query="select Id, ActualScheduleTime, AdapterStatus,
AssociatedReference, Beneficiary, BeneficiaryBank, Branch,
ChannelId, Correspondent, Counterparty, CRAccountName,
CRNumber, CRSortCode, Currency, CurrentId,
CustomerReference, DealDate, Dealer, DestinationAccount,
DRAccountName, DRNumber, DRSortCode, EventType, Expired,
ExpiryDateTime, ExternalReference, MatchedAmount, MatchId,
MatchIgnored, MatchType, MaturityEvent, MTType,
OrderingInstitution, OriginalAmount, OriginalScheduleTime,
OriginatingSystem, Originator, OriginatorReference,
PartialMatchAllwd, PayReceive, PreviousChannel,
ProcessedDateTime, Product, RemainingAmount, RemittingBank,
ReRouted, ResponsibleUnit, Reversed, SourceSystemInfo,
Status, StreamMovementType, SystemCounterparty,
TransactionType, ValueDateTime, W_ValueDate from
Transactions where trunc(ValueDateTime) = undate('${p1}')"
fieldmap="TransactionsFieldMap" />

<!-- Events table field map -->
<FieldMapDefinition id="EventsFieldMap">
<Column targetFieldName="2887" />
<Column targetFieldName="2889" />
<Column targetFieldName="2891" />
<Column targetFieldName="2893" />
<Column targetFieldName="2895" />
<Column targetFieldName="2897" />
<Column targetFieldName="2899" />
<Column targetFieldName="2901" />
<Column targetFieldName="2903" />
<Column targetFieldName="2905" />
<Column targetFieldName="2907" />
<Column targetFieldName="2909" />
<Column targetFieldName="2911" />
<Column targetFieldName="2913" />
<Column targetFieldName="2915" />
<Column targetFieldName="2917" />
<Column targetFieldName="2919" />
<Column targetFieldName="2921" />
<Column targetFieldName="2923" />
<Column targetFieldName="2925" />
<Column targetFieldName="2927" />
<Column targetFieldName="2929" />
<Column targetFieldName="2931" />
<Column targetFieldName="2933" />
<Column targetFieldName="2935" />
<Column targetFieldName="2937" />
<Column targetFieldName="2939" />
<Column targetFieldName="2941" />
<Column targetFieldName="2943" />
<Column targetFieldName="2945" />
<Column targetFieldName="2947" />
<Column targetFieldName="2949" />
<Column targetFieldName="2951" />
<Column targetFieldName="2953" />
<Column targetFieldName="2955" />
<Column targetFieldName="2957" />
```

```
<Column targetFieldName="2959" />
<Column targetFieldName="2961" />
<Column targetFieldName="2963" />
<Column targetFieldName="2965" />
<Column targetFieldName="2967" />
<Column targetFieldName="2969" />
<Column targetFieldName="2971" />
<Column targetFieldName="2973" />
<Column targetFieldName="2975" />
<Column targetFieldName="2977" />
<Column targetFieldName="2979" />
<Column targetFieldName="2981" />
<Column targetFieldName="2983" />
<Column targetFieldName="2985" />
<Column targetFieldName="2987" />
<Column targetFieldName="2989" />
<Column targetFieldName="2991" />
<Column targetFieldName="2993" />
<Column targetFieldName="2995" />
<Column targetFieldName="2997" />
<Column targetFieldName="2999" />
<Column targetFieldName="3001" />
<Column targetFieldName="3003" />
<Column targetFieldName="3005" />
<Column targetFieldName="3007" />
<Column targetFieldName="3009" />
<Column targetFieldName="3011" />
<Column targetFieldName="3013" />
<Column targetFieldName="3015" />
</FieldMapDefinition>

<MigrateStream id="Events"
targetTable="262"
method="query"
host="foobar"
query="select UniqueId, ActualScheduleTime, AdapterStatus,
AssociatedReference, Beneficiary, BeneficiaryBank, Branch,
CalculatedId, CaptureDateTime, CaptureOperator, ChannelId,
ActualScheduleTime, Correspondent, Counterparty,
CounterpartyAlias, CRAccountName, CRNumber, CRSortCode,
Currency, CurrentId, CustomerReference, DealDate, Dealer,
DestinationAccount, DRAccountName, DRNumber, DRSortCode,
EventDateTime, EventType, Expired, ExternalReference,
FrontOfficeId, InterfaceId, MaturityEvent, MTType,
OrderingInstitution, OrderingInstitutionAlias,
OriginalAmount, OriginalId, OriginalScheduleTime,
OriginatingSystem, Originator, OriginatorAlias,
OriginatorReference, PartialMatchAllwd, PreviousChannel,
PreviousId, ProcessedDateTime, Product, RemittingBank,
ReRouted, ResponsibleUnit, Reversed, Rollable,
RollableFromId, SourceSystem, SourceSystemEventInfo,
SourceSystemInfo, StatusWeight, StreamMovementType,
SystemCounterparty, SystemCounterpartyAlias, Text,
TransactionType, ValueDateTime from Events where
trunc(ValueDateTime) = undate('${p1}')"
fieldmap="EventsFieldMap" />

<OlapServices id="OlapServicesList">
  <Service host="olapmachine" port="11106" />
</OlapServices>

</PlatformMigration>
```

Copyright

See Also

sp(1)

Name

sp_kdbin — reads data from a kdb+tick database table into an Sybase Aleri Streaming Platform stream.

Synopsis

`sp_kdbin` -H *`[kdbhost:]kdbport`* -p *`[host:]port`* -q *`source`* -s *`stream`* [*`OPTION`*...]

Description

The **sp_kdbin** adapter reads data from a kdb or kdb+tick database into a stream in the Sybase Aleri Streaming Platform. The adapter can read either queried or streaming data, based on a configuration parameter.

By default, the adapter matches the field names (in a case-insensitive manner) to decide the mapping between the source kdb+tick table and the target stream. You also have the option of explicitly specifying the mapping.

Required Arguments

`-H` *`[kdbhost:]kdbport`*
: Specifies the port number, or the host name and port number, on which KDB is listening. The default host name is localhost.

`-p` *`[host:]port`*
: Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

`-q` *`source`*
: Specifies the kdb+tick table when running in streaming mode. When running in non-streaming mode, specifies a valid query string.

`-s` *`stream`*
: Specifies the target stream: the stream to which the data being read is published.

Options

`-a`
: Use asynchronous mode transmission; where the adapter does not wait for acknowledgment from the Sybase Aleri Streaming Platform that it received the data. This option is necessary when using a hot spare configuration to ensure that both the primary and the hot spare receive the data.

`-b` *`blocksize`*
: Specify how many records to put in a block of data for transmission. The default is 64. A higher value may increase throughput but it will also increase latency. A block may contain fewer records than specified if there is not enough data available.

`-c` *`user[:password]`*
: Pass authentication credentials to the Sybase Aleri Streaming Platform. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-d`

Output debug messages.

`-e`

Use encrypted OpenSSL sockets for all communications between the adapter and the Sybase Aleri Streaming Platform (which requires that it be started in encrypted mode). When this option is not present, no encryption occurs.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller
2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-g` *`gatewayhost`*

Use the specified gateway host. Ignore the host name returned by the Sybase Aleri Streaming Platform.

`-h`

Print detailed help.

`-k` *`privateRsaKeyFile`*

Authenticate using the RSA private key file mechanism instead of a password. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-I` *`interval`*

Specifies the number of seconds to wait before running the supplied query again when running in non-streaming mode. A value of `0` indicates that the query should only be run once; no polling is performed.

`-M` *`mapping`*

Specifies a mapping between the column name in the target stream and the column name in the kdb+tick database table. The mapping is a colon separated series of SPColumn=KDBColumn statements. If this parameter is not provided, the connector will absorb data only for those columns where the target stream column name matches the source table column name (in case-insensitive manner).

`-m`

If this option is not specified, the adapter connects to a kdb+tick database and reads in streaming data. If it is specified, the adapter executes the supplied database query and feeds the result to the Sybase Aleri Streaming Platform.

`-T` *`attempts`*

Specifies the number of times to attempt to re-connect to the kdb or kdb+tick database if the connection breaks during operation. The default is 1.

`-t`

Use transaction blocks. The default is to use envelopes. This improves performance, but will cause all records in a block to be rejected when one record fails.

`-u` *`user[:password]`* Pass authentication credentials to the KDB database.

Examples

To execute a basic streaming mode query that reads data from a KdbTrades table in a kdb+tick database on the server altair, where KDB is listening on port 9200, and writes it to the SpTrades stream in the Sybase Aleri Streaming Platform on the local server where the Command and Control interface is on port 1190,

```
sp_kdbin -p 1190 -H altair:9200 -q KdbTrades -s SpTrades
```

To execute the same query, explicitly mapping fields in the kdb+tick database to columns in the Sybase Aleri Streaming Platform stream,

```
sp_kdbin -p 1190 -H altair:9200 -q KdbTrades -s SpTrades -M SpId=KId:SpSymbol=KSymbol:SpPrice=KPrice:Sp
```

To execute a pull mode operation that issues the specified query every 5 seconds to a kdb+tick database on the server altair, where KDB is listening on port 9200, and writes data to the Sybase Aleri Streaming Platform, on the server ceres, where the Command and Control interface is on port 1221,

```
sp_kdbin -p ceres:1221 -H altair:9200 -q 'select Id, Symbol, Price, Count from KdbTrades'  -s SpTrades
```

Copyright

See Also

sp(1)

Name

sp_kdbout — feeds streaming data from the Sybase Aleri Streaming Platform to a kdb+tick database table.

Synopsis

`sp_kdbout` -H *`[kdbhost:]kdbport`* -p *`[host:]port`* -q *`source`* -s *`table`* [*`OPTION`*...]

Description

The **sp_kdbout** adapter feeds streaming data from the Sybase Aleri Streaming Platform to a kdb+tick database table.

By default, the adapter matches the field names (in a case-insensitive manner) to decide the mapping between and the Sybase Aleri Streaming Platform stream and the kdb+tick table. You also have the option of explicitly specifying the mapping.

Required Arguments

`-H` *`[kdbhost:]kdbport`*
Specifies the port number, or the host name and port number, on which KDB is listening. The default host name is localhost.

`-p` *`[host:]port`*
Specifies the port number, or the host name and port number, of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. The default host name is localhost.

`-q` *`source`*
Specifies either the name of a stream on the Sybase Aleri Streaming Platform or a valid SQL query to retrieve the data to write to the kdb+tick table.

`-s` *`table`*
Specifies the name of the kdb+tick table to which the data will be written.

Options

`-a`
Use asynchronous mode transmission; where the adapter does not wait for acknowledgment from the Sybase Aleri Streaming Platform that it received the data. This option is necessary when using a hot spare configuration to ensure that both the primary and the hot spare receive the data.

`-B`
Use droppable subscriptions. The Sybase Aleri Streaming Platform will drop the subscription if the adapter cannot keep up with the data.

`-b` *`batchsize`*
This option allows you to set the maximum number of records to include in a single batch write to kdb+tick. The default is 5000.

`-c` *`user[:password]`*
Pass authentication credentials to the Sybase Aleri Streaming Platform. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option.

If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-d`

Output debug messages.

`-e`

Use encrypted OpenSSL sockets for all communications between the adapter and the Sybase Aleri Streaming Platform (which requires that it be started in encrypted mode). When this option is not present, no encryption occurs.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-h`

Print detailed help.

`-I` *`fieldnames`*

Specify a comma-separated list of kdb field names whose values will be ignored. These fields are included in the message, but are always populated with NULL.

`-K`

If specified, the subscription from the Sybase Aleri Streaming Platform preserves the transaction boundaries. The default is False.

`-k` *`privateRsaKeyFile`*

Authenticate using the RSA private key file mechanism instead of a password. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-L` *`interval`*

Use pulsed subscribe when connecting to the Sybase Aleri Streaming Platform. The pulse *`interval`* is specified in seconds.

`-l`

Use “lossy” subscribe.

`-M` *`mapping`*

Specifies a mapping between the column name in the target stream and the column name in the kdb+tick database table. The mapping is a colon separated series of SPColumn=KDBColumn statements. If this parameter is not provided, the connector will absorb data only for those columns where the target stream column name matches the source table column name (in case-insensitive manner).

`-m`

Write the data to the table using the “upsert” operation. If this option is not used, the adapter works in streaming mode and uses the “u.upd” operation to write data to the kdb+tick database.

`-O` *`fieldnames`*
Specify a comma-separated list of kdb field names to omit from the message. Unlike ignored fields which are part of the message but always NULL, these fields are not included in the message.

`-R`
Subscribe with shine through (if possible) so that previously received information is retained for any field(s) for which the update contains no new data.

`-T` *`attempts`*
Specifies the number of times to attempt to re-connect to the kdb or kdb+tick database if the connection breaks during operation. The default is 1.

`-t`
Send data to kdb+tick in transaction blocks. The base data existing in the stream at the time of connection is not sent.

`-u` *`user[:password]`*
Pass authentication credentials to the KDB database.

Examples

To subscribe to the SpTrades stream in the Sybase Aleri Streaming Platform on the server ceres, where the Command and Control interface is on port 1221, and stream the data out to KdbTrades in the kdb+tick database on the server altair, where KDB is listening on port 9200,

```
sp_kdbout -p ceres:1221 -H altair:9200 -q SpTrades -s KdbTrades
```

To populate fields XXX and YYY in a kdb+tick table may with NULL (because they have no corresponding data in the Sybase Aleri Streaming Platform),

```
sp_kdbout -p ceres:1221 -H altair:9200 -q SpTrades -s KdbTrades -I XXX,YYY
```

Sometimes a table in kdb+tick may compute fields (for example, XXX and YYY) which should not be specified in the data update message. Doing so usually results in a "length" error in kdb+tick. To omit these fields from the update message altogether,

```
sp_kdbout -p ceres:1221 -H altair:9200 -q SpTrades -s KdbTrades -O XXX,YYY
```

Copyright

See Also

sp(1)

Name

sp_stream2olap — migrate a data stream to V-OLAP

Synopsis

`sp_stream2olap` -F *`xsdfile`* -f *`configfile`* -l *`logdir`* -o *`outpath`* -p *`[host:]port`* [*`OPTION`*...]

Description

The **sp_stream2olap** adapter is used to migrate streaming data from the Sybase Aleri Streaming Platform to the V-OLAP system. The adapter establishes a connection to a running instance of the Sybase Aleri Streaming Platform Based on configuration it will then create subscriptions to one or more streams. In addition to regular subscriptions, **sp_stream2olap** supports projected subscriptions which are based on SQL queries. Subscriptions can also be persisted. The **sp_stream2olap** adapter receives data from the Sybase Aleri Streaming Platform and writes it out to disk in a format readable by V-OLAP and notifies any running OLAP instances of the new data.

In addition to streaming, **sp_stream2olap** supports a snapshot mode. In this mode, **sp_stream2olap** subscribes to the configured streams, reads the base data (the data already stored in the stream at the time of connection) and exits.

The adapter is driven by a combination of parameters read from an XML configuration file and command line options.

Required Arguments

`-F` *`sp_stream2olap.xsd`*

Specify the XSD Schema file for validating the XML configuration file. You must enter the full pathname of the XSD schema file.

`-f` *`configuration_file`*

Specify the XML configuration file which contains a description of the source data along with instructions on how it is to be migrated.

`-l` *`logdir`*

Specify the directory where the log file, `sp_stream2olap.log`, is saved.

`-o` *`outpath`*

Specify the full pathname of the folder where the V-OLAP vector store will be written. If starting in “create” mode, this folder must be empty.

It is important to specify a directory on a disk that has a low probability of running out of disk space or crashing.

`-p` *`[hostname:]port`*

Specify the *`port`*, and optionally the *`hostname`*, of the command and control interface within a running instance of the Sybase Aleri Streaming Platform. If you do not specify a hostname, the default is localhost.

Options

`-C`

sp_stream2olap creates a fresh vector store using the stream definitions retrieved from the Sybase Aleri Streaming Platform. When this option is specified, the output folder specified using the `-o` option must not contain the V-OLAP table folders that will be written to by **sp_stream2olap**.

When producing vector store file names, the **sp_stream2olap** application appends a trailing slash onto the end of the value that the outpath attribute specifies.

`-c` *`user[:password]`*

If you are using an authentication method that requires credentials (Kerberos, PAM, or RSA), this option passes those authentication credentials to the Sybase Aleri Streaming Platform. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise it is closed immediately.

The specific values for this option depend on the type of authentication being used by the Sybase Aleri Streaming Platform. If no authentication is being used, then nothing should be specified. For PAM authentication, user and password refer to the PAM account credentials. For RSA and GSSAPI, specify only the user and omit the password.

`-D`

Run in daemon mode. In this mode, **sp_stream2olap** spawns a daemon process. The process id of the daemon is output to the console. This id can be used to signal the daemon. Refer to the documentation below for details.

`-d` *`n`*

Generate debug messages at the specified debug level: from 1 to 4.

`-e`

Encrypt the socket connection between **sp_stream2olap** and the Sybase Aleri Streaming Platform.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required.

`-h`

Enter this option to display the usage message for this command.

`-k` *`privateRSAKeyFile`*

Perform authentication using the RSA private key file mechanism instead of password authentication. The *`privateRSAKeyFile`* must specify the absolute path filename of the private RSA key file. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Sybase Aleri Streaming Platform must be started with the `-k` option.

`-O` *`hostname:port[;hostname:port]`*

Specify one or more V-OLAP services to connect to and notify when data is written. Multiple services are separated by semicolons. The services specified on the command line override any definitions in the XML configuration file.

`-s`

Run in snapshot mode. In this mode, **sp_stream2olap** reads just the base data in all the configured streams and then exits.

`parameter=value`...

sp_stream2olap allows placeholders in the SQL query that it will use to subscribe to a stream. Specify the values for these placeholders at the end of the command line argument list. This is a set of query substitution parameter name=value pairs. The name of a substitution parameter must start with a lower case "p" character, followed immediately by one or more digits, followed immediately by an equal '=' sign, that is then followed immediately by the value. Within the XML configuration file, the SQL statements may look something like this:

```
select BalanceType from BalanceMovements where f1 = ${p1}
```

At runtime, all of the ${p1} place holders within SQL queries will be replaced with the value assigned to the p1 parameter on the command line.

Configuration

sp_stream2olap migration is driven by an XML configuration file that must conform to the schema shipped with the adapter in `sp_stream2olap.xsd`. The configuration file lists the streams that are to be migrated and describes the output vector store files. Below is a simple configuration file that illustrates all supported options.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlatformMigration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- Sample XML configuration file for the Aleri Platform Data Migration Adapter -->
    <Environment id="RTRisk_Export" source="platform"
        blockSize="5000"
        online="optional" />
    <!-- Transactions table field map -->
    <FieldMapDefinition id="RTRisk_export_map">
      <Column targetFieldName="TradeId"/>
      <Column targetFieldName="Side"/>
      <Column targetFieldName="Party"/>
      <Column targetFieldName="Instrument"/>
      <Column targetFieldName="Venue"/>
      <Column targetFieldName="TradeTime"/>
      <Column targetFieldName="Size"/>
      <Column targetFieldName="DealValue"/>
      <Column targetFieldName="Trader"/>
      <Column targetFieldName="RiskUSD"/>
      <Column targetFieldName="RiskEUR"/>
      <Column targetFieldName="RiskLimit"/>
      <Column targetFieldName="RiskConcentration"/>
      <Column targetFieldName="EventTime"/>
    </FieldMapDefinition>
    <MigrateStream id="DealRiskDelta"
        targetTable="risk_facts"
        method="projection"
        query="select sequenceNumber, opcode,
TradeId,Side,Party,Instrument,Venue,TradeTime,Size,DealValue,Trader,RiskUSD,RiskEUR,
RiskLimit,RiskConcentration,TradeTime from DealRiskDelta_log where SeqNbr &gt; ${p1}
and SeqNbr &lt; (1 + ${p2})"
          host="localhost"
        controlstream="DealRiskDelta_truncate"
        session="10"
          fieldmap="RTRisk_export_map" />
    <!-- Events table field map -->

    <OlapServices id="OlapServicesList">
        <Service host="chidt025ux" port="11106" />
    </OlapServices>
</PlatformMigration>
```

An **sp_stream2olap** XML configuration file contains one top most element: PlatformMigration. PlatformMigration can contain the following nodes: Environment, OlapServices, FieldMapDefinition, and MigrateStream.

Environment

This entity specifies certain global parameters for **sp_stream2olap**. Its attributes are:

`source`

Mandatory string attribute. Currently the only supported value is “platform”.

`blockSize` This optional integer attribute specifies the maximum number of Sybase Aleri Streaming Platform records to batch together before writing the data to disk. The default value is 64.

When migrating live data, **sp_stream2olap** keeps batching records until the record count reaches the specified `blockSize`. Or, if there is no immediate data being sent by the Sybase Aleri Streaming Platform. all data read so far is written to disk instead of waiting indefinitely.

`online` This optional string attribute specifies how **sp_stream2olap** should deal with OlapServices. It can have one of three values:

none **sp_stream2olap** does not look for or notify any OLAP services of the data it has written.

mandatory **sp_stream2olap** will try to connect to configured OLAP services. If it fails, **sp_stream2olap** treats that as a fatal error and exits. If a connection to OLAP service is lost while it is migrating, this is also treated as a fatal error.

optional **sp_stream2olap** will try to connect to OLAP but will not exit if one is not found. Similarly, a lost connection while running will not cause **sp_stream2olap** to exit.

FieldMapDefinition

A FieldMapDefinition node describes the structure of a vector data store for an OLAP table. It has a set of ordered Column elements. The order in which these Column elements are defined should correspond to the order of the columns in the stream being subscribed to in the Sybase Aleri Streaming Platform. If using an SQL query, these should correspond to the order of the field names in the SQL query statement.

Id A string that uniquely identifies this FieldMapDefinition node.

Column

A column element corresponds to a field in a table definition.

targetFieldName This string attribute specifies the name to use for the column when writing data to disk. **sp_stream2olap** uses this to name the vector file that will contain data for the field.

targetRefField Optional boolean attribute. If set to true, indicates that the column data should be stored as a reference type in the vector data store. This attribute only applies to symbol (character) data fields.

MigrateStream

A MigrateStream entity describes the migration between an Sybase Aleri Streaming Platform stream and an OLAP table. This includes the mappings and operation modes.

id

This mandatory string attribute uniquely identifies this MigrateStream node.

targetTable

In the historical vector store, each table is represented by a directory. This mandatory string attribute specifies the name for that directory. **sp_stream2olap** creates these directories under the “outpath” folder specified in the command line.

fieldmap

This mandatory string attribute refers to an existing FieldMapDefinition node id that is used to map the columns from the stream being read.

method

This mandatory string attribute can be set to either “subscription” or “projection”. If set to subscription, the connection to the stream is made using a regular subscription. If set to projection, the connection is made via a SQL query which is used to filter the data from the stream.

query

If the method attribute is set to “subscription”, this refers the stream name in the Sybase Aleri Streaming Platform from which to read data. If method is set to “projection”, this must specify a valid SQL query. Care must be taken in naming the fields in the query since these must map one-to-one to the columns in the related FieldMapDefinition.

host

This mandatory string attribute contains the name of the host machine exactly as it appears in the definition of the Fact Table partition for the Virtual Cube in the OLAP metadata.

controlstream

This attributes names the stream in the Sybase Aleri Streaming Platform model that is used to clear published records.

session

This integer attribute can be any positive integer greater than zero, as long as it is different from the session of any other MigrateStream node that maybe writing to the same physical vector store table. This will include MigrateStreams across all instances of **sp_stream2olap** that may be running.

priority

This mandatory integer attribute assigns a priority to the MigrateStream node. The priority attribute plays a role if multiple MigrateStream nodes are defined. **sp_stream2olap** uses the priority value to decide which stream to write to the disk first when there is simultaneous data from multiple streams. A lower value indicates higher priority.

The controlstream and session attributes are used for persistent subscribes. Please refer to the section on persistent subscribes for details on how to set this up.

OlapServices

The OlapServices entity contains a list of host and port pairs, each representing a running V-OLAP. Depending on the value of the online attribute of the Environment entity, **sp_stream2olap** will notify each V-OLAP service when a block of data has been successfully written to disk.

Usage Notes

sp_stream2olap is started from the command line with various options. It can run either in console or daemon mode. In console mode, **sp_stream2olap** provides a command prompt. At this prompt, you can enter the following commands to control **sp_stream2olap**:

quit

Causes **sp_stream2olap** to exit

connect olap [host port]

Causes **sp_stream2olap** to reconnect to all configured OLAP services. If a specific host and port is supplied, **sp_stream2olap** tries to connect to that service even if it is not one that has been specified in the configuration file.

show olap Displays configured OLAP services and the connection status.

SQL Substitution Parameters

sp_stream2olap allows SQL queries to contain named placeholders. If the SQL query does contain parameters, values for these parameters must be specified in the command line. For example, if a query is structured as follows:

```
select * from Trades where TradeId < ${p1} and TradeId > ${p2}
```

You must specify the values for the named parameters when you start **sp_stream2olap**. For example,

```
sp_stream2olap -p 3241 -f dm.xml -o output -l log p1=0 p2=10000
```

Persistent Subscriptions

sp_stream2olap supports persistent subscriptions to the Sybase Aleri Streaming Platform. In persistent subscription mode, a client and the Sybase Aleri Streaming Platform cooperate to keep track of records that have been successfully read by the client. This information is persisted. In the event of a crash of either the client or the Sybase Aleri Streaming Platform, the subscription can be resumed from the point of the last successful read.

In order to work, a model needs to be modified to support a persistent subscription pattern. Please refer to the *Sybase Aleri Streaming Platform Authoring Guide* for a detailed description of this pattern. The Aleri Studio can be used to make these modifications automatically. As part of this modification, the Aleri Studio creates two additional streams for each stream that is to be subscribed to persistently. These are named by appending the extensions `_log` and `_truncate` to the original stream name. For example, if the stream that needs to be subscribed to persistently is named `Trades`, the Aleri Studio will create `Trades_log` and `Trades_truncate`.

The XML configuration file also needs to change. The query attribute of the MigrateStream node that will be using persistent subscriptions now has to refer to `Trades_log` as the stream to read instead of the original `Trades`. If the subscription is projected (that is, it has an SQL query), then it too needs to change to add two additional columns: `sequenceNumber` and `opcode` as the first two fields on the query. For example, if the original SQL query was

```
select TradeId,
TradeTime,
Value from Trades
```

It now needs to be

```
select sequenceNumber,
opcode,
TradeId,
TradeTime,
Value from Trades_log
```

In addition to change in the “query” attribute, two additional attributes need to be specified: “controlstream” and “session”. The “controlstream” attribute should refer to the second stream that the Aleri Studio creates. In our example it is “Trades_truncate”. The “session” attribute is an integer. While most of the time this can be any value greater than zero, it is important to set this value properly if the installation has multiple MigrateStream nodes pointing to the same physical vector store table. This holds true even if the migration is being done in multiple instances of **sp_stream2olap**. In this case, it is important to assign a different integer value to each MigrateStream node that shares the same physical output location. **sp_stream2olap** uses this value to distinguish recovery information.

Snapshot mode

Snapshot mode is set with the `-s` command line option. In this mode, **sp_stream2olap** will exit once the base data from all configured streams has been read and written to disk. Base data is the data that already exists in the stream at the time the connection is made.

In order to be useful, it is important to make sure that the Sybase Aleri Streaming Platform is quiesced.

Daemon mode

This mode is set with the `-D` command line option. In this mode, **sp_stream2olap** spawns a daemon process to continue the migration and outputs the process id of the daemon. The process id can be used to signal the **sp_stream2olap** daemon process. This can be useful if any OLAP services go down and **sp_stream2olap** loses the connection.

Currently **sp_stream2olap** recognizes two signals: SIGTERM and SIGUSR1. SIGTERM causes **sp_stream2olap** to exit cleanly. SIGUSR1 causes **sp_stream2olap** to try to reconnect to all configured OLAP services.

Run the **sp_stream2olap** in daemon mode and save the process id for use in sending signals to it.

```
> SPID=`sp_stream2olap -p 22000 -o output -l log -D -f config.xml`
```

Then, to have **sp_stream2olap** reconnect to OLAP services:

```
> kill -10 $SPID
```

Or, to terminate the program:

```
> kill -15 $SPID
```

The numbers `10` and `15` in the preceding commands are the signal numbers for the SIGUSR1 and SIGTERM signals, respectively. Entering **`kill -l`** at the command line prompt will get you a complete list of signals and signal numbers.

Recovery

The **sp_stream2olap** utility encapsulates each table migration within a single recoverable transaction. Either all of the table columns are updated or none of them are. In case of a data migration failure, this mechanism preserves the target vector store area, leaving it in a consistent state. Each table being handled by **sp_stream2olap** has its own recovery file.

The mechanism used to implement recovery is a WAL (Write-Ahead-Log). Before writing out a fresh batch of data, the header of each vector store column file is written out to the recovery file. After the vector store headers have been successfully backed up in the recovery file, the adapter attempts to append the Sybase Aleri Streaming Platform data onto the corresponding vector store. If all goes well, and all vector store files have been successfully appended to, the adapter will truncate the recovery file. If there is a non-fatal error at any point in the migration attempt, the adapter will detect the error and attempt to recover/rollback the vector store files to the state that they were in prior to the batch before exiting. If a successful recovery is accomplished, **sp_stream2olap** will truncate the recovery file.

If a fatal error occurs, and the adapter aborts in the middle of a migration attempt, the next time the adapter is started up, it will detect if a particular table has a viable recovery file. If it has, the adapter will attempt to recover/rollback the corresponding vector store table using the entries in the recovery file. Upon a successful start-up recovery/rollback, **sp_stream2olap** will truncate the recovery file and proceed with the migration.

Configuration Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="SessionId">
      <xs:restriction base="xs:integer">
          <xs:minInclusive value="1"/>
      </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="AsapObject">
    <xs:attribute name="name"          type="xs:string" use="optional"/>
    <xs:attribute name="documentation" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:element name="PlatformMigration" type="Container"/>
  <xs:complexType name="Container">
    <xs:complexContent>
      <xs:extension base="AsapObject">
        <xs:sequence>
           <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="Environment"     type="Environment" />
             <xs:element name="Container"       type="Container" />
             <xs:element name="MigrateStream"   type="MigrateStream" />
             <xs:element name="OlapServices"    type="OlapServices" />
             <xs:element name="FieldMapDefinition"   type="FieldMapDefinition">
               <xs:key name="UniqueColumnConstraint2">
                 <xs:selector xpath="Column"/>
                 <xs:field xpath="@targetFieldName"/>
               </xs:key>
             </xs:element>
           </xs:choice>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Environment">
    <xs:complexContent>
      <xs:extension base="AsapObject">
        <xs:attribute name="id" type="xs:ID" use="required"/>
            <xs:attribute name="source" use="required" >
          <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="platform"/>
                </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
            <xs:attribute name="online" use="optional" default="none">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="none"/>
                    <xs:enumeration value="mandatory"/>
                    <xs:enumeration value="optional"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="blockSize" type="xs:integer" use="optional" default="64" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FieldMapDefinition">
    <xs:complexContent>
      <xs:extension base="AsapObject">
        <xs:sequence>
```

```
          <xs:element name="Column" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
                  <xs:attribute name="targetFieldName" type="xs:string" use="required"/>
                  <xs:attribute name="targetRefField" type="xs:boolean" use="optional" default="false"/
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MigrateStream">
    <xs:complexContent>
      <xs:extension base="AsapObject">
        <xs:attribute name="id" type="xs:ID" use="required"/>
            <xs:attribute name="targetTable" type="xs:string" use="required"/>
            <xs:attribute name="fieldmap" type="xs:IDREF" use="required" ecore:reference="FieldMapDefin
            <xs:attribute name="method"  use="required" >
          <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="subscription"/>
                      <xs:enumeration value="projection"/>
                </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="query" type="xs:string" use="required"/>
        <xs:attribute name="controlstream" type="xs:string" use="optional"/>
        <xs:attribute name="session" type="SessionId" use="optional"/>
        <xs:attribute name="host" type="xs:string" use="optional" default=""/>
        <xs:attribute name="priority" type="xs:integer" use="optional" default="0"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="OlapServices">
    <xs:complexContent>
      <xs:extension base="AsapObject">
        <xs:sequence>
          <xs:element name="Service" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="host" type="xs:string" use="required"/>
                  <xs:attribute name="port" type="xs:integer" use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Examples

To run **sp_stream2olap**, logging in to port 9900 on a server named ohio, using the configuration file `/home/aleri/config.xml`, writing the output to `/home/aleri/output`, and saving the logfile `sp_stream2olap.log` in the `/var/logs` directory:

```
sp_stream2olap -p ohio:9900 -f /home/aleri/config.xml -o /home/aleri/output -l /var/logs
```

Copyright

Name

sp_subscribe — A sample subscription client for the Sybase Aleri Streaming Platform.

Synopsis

`sp_subscribe` -c *`user:passwd`* -p *`[hostname:]port`* [*`OPTION`*...]

Description

The **sp_subscribe** client connects to an instance of the Sybase Aleri Streaming Platform via the Command and Control and Gateway interfaces and subscribes to transaction streaming data. The received records are converted to XML (or optionally delimited format) and written to the standard output.

Options

`-A` *`connectionHandle`*
: Instead of subscribing itself, subscribe the already existing connection with handle *`connectionHandle`* to the specified streams. This **sp_subscribe** will exit after processing the request.

`-B`
: Specifies that the connection should be dropped if it supplies data faster than **sp_subscribe** can absorb it.

`-b`
: Indicates that the machine architecture on which the server supplying us with data is running has the reverse byteorder of the machine architecture on which **sp_subscribe** is running.

`-c` *`user[:password]`*
: Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

  This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-D` *`connectionHandle`*
: Instead of subscribing itself, unsubscribe the already existing connection with handle *`connectionHandle`* from the specified streams. This **sp_subscribe** will exit after processing the request.

`-d` *`separator`*
: Put the subscribe client into delimited output mode and use the specified separator as the delimiting character.

`-e`
: Negotiate encrypted OpenSSL sockets for all communication with the Sybase Aleri Streaming Platform (which requires that it be started in encrypted mode).

`-G`
: Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

  You can obtain multiple valid Kerberos tickets by:

  1. authenticating using a Kerberos server other than your current domain controller

2. logging in to one user account and using a different one to connect to the Sybase Aleri Streaming Platform

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

`-g` *`gateway_host`*

Use the specified gateway host rather than the host returned from the `get_gateway()` call.

`-H`

Subscribe to the Sybase Aleri Streaming Platform heartbeat messages instead of data.

`-h`

Print detailed help.

`-i` *`streamId[,...]`*

Specify one or more streams to subscribe to, using the integer handle of each stream.

`-K`

Subscribe to the original transaction blocks generated within the Sybase Aleri Streaming Platform. This option prevents those original transaction blocks from being coalesced into more optimal blocks.

`-k` *`privateRsaKeyFile`*

Authentication is performed using the RSA private key file mechanism instead of password authentication. The privateRsaKeyFile must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-L` *`period`*

Specifies a pulsed subscription, where records are collected for the specified period in seconds, and then delivered.

`-l`

Put the subscribe client into lossy mode, where data is shed (lost) if the subscribe client cannot keep up with the streaming data produced by the Sybase Aleri Streaming Platform.

`-M` *`number`*

Specify the number of multiple connections to establish (all for the same streams).

`-m` *`name`*

Set a symbolic tag name for the connection. This allows to look up the connection easily in the Aleri_Clients metadata stream. The name will be stored in the field “conn_tag”. To kill the connections by tag name, use the **sp_cli** command "kill every".

`-P` *`precision`*

Set the number of decimal places in output for DOUBLE (default 6).

`-p` *`[hostname:]port`*

Specify the port or the hostname and port of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform.

`-Q` *`“SQL statement”`*

Specifies an SQL “select” statement to apply to outbound records. The SQL statement can use any of the features of the SQL syntax given in the *Guide to Programming Interfaces*.

The outbound records come marked with insert, update, and delete, even when the SQL statement might not include the key columns of the stream. To give insert/update/delete meaning in these cases, you can specify a special column ALERI_SEQNO. For example, the statement “select *, ALERI_SEQNO from Vwap order by Price” fills in values for the ALERI_SEQNO column automatically. With “order by”, the ALERI_SEQNO reflects the ordering of the rows in the output set.

`-R` Subscribe with shine through (if possible) so that when an update contains no new data for some field(s) previously received information for that field(s) will be retained.

`-S` Take a snapshot of the table: receive the initial state of the stream and exit immediately afterward.

`-s` *`streamName[,...]`* Specify one or more streams to subscribe to, using the logical name of each stream.

`-T` Print out block begin (namely <block>) and block end (namely </block>) around transaction blocks.

`-t` Put the subscribe client into transaction-only mode. It will receive transactional updates to streams upon connection to the Sybase Aleri Streaming Platform and not the initial state of the streams.

`-v` Print the service XML entries, not only the data.

`-W` *`baseDrainTimeout`* Set the time limit, in milliseconds, that this client has to read all base data from the subscribe stream before being dropped. When this parameter is not specified, the default value is 8,000 milliseconds or 8 seconds.

`-X` Exit the process after all the subscriptions exit, even if the Sybase Aleri Streaming Platform does not drop the connection.

`-z` Set the internal queue size of the output gateway connection. This queue front ends the connected socket, buffering data to be delivered to the client. The default queue size is 8192 records.

Examples

To subscribe to two streams named PreprocessorTransactions and DebitMovements, printing all stream data in XML format on stdout:

```
sp_subscribe -c user:pass \
  -s PreprocessorTransactions,DebitMovements -p 11180
```

To subscribe to two streams named PreprocessorTransactions and DebitMovements, printing all stream data in pipe-separated format on stdout:

```
sp_subscribe -c user:pass -d "|" \
  -s PreprocessorTransactions,DebitMovements -p 11180
```

To subscribe to one stream named PreprocessorTransactions whose data is generated by a server running on a machine named HOST (which has differing byteorder than the machine that subscribe is running on) and print all stream data in pipe-separated format on stdout:

```
sp_subscribe -c user:pass -d "|" \
  -s PreprocessorTransactions -p HOST:11180
```

To subscribe to one stream named baseInput and apply a SQL statement:

```
sp_subscribe -c user:pass -Q \
  "select intData_1, 10*intData_1+dblData_1
    from baseInput where intData_1 > 20" \
  -p HOST:11180
```

Copyright

See Also

sp(1), sp_convert(1), sp_upload(1)

Bugs

See the documentation for known issues.

Name

sp_upload — A sample record upload client for the Sybase Aleri Streaming Platform

Synopsis

`sp_upload` [*`OPTION`*...]

Description

The **sp_upload** utility reads binary records from the standard input and forwards them to a running instance of the Sybase Aleri Streaming Platform via the Gateway interface.

The format of the data is zero or more occurrences of <Stream Handle><Raw Binary Record>. <Stream Handle> is a uint32_t indicating the destination stream for the record. The <Raw Binary Record> format is documented in the *Guide to Programming Interfaces*. This tool is typically used at the end of a pipeline with the **sp_convert** tool.

Options

`-b`

Set byteswap mode. The raw records fed into **sp_upload** (via **sp_convert** -b) and the server to which **sp_upload** is sending data, BOTH have different byte order than the architecture the **sp_upload** client is running on (the byte order of the data must always match the byte order of the server).

`-c` *`user[:password]`*

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`-d` *`num`*

Insert a delay of *`num`* microseconds between records or transaction blocks.

`-e`

Negotiate encrypted OpenSSL sockets for all communication with the Sybase Aleri Streaming Platform (which must be started in encrypted mode to use this option).

`-f` *`num:SQL`*

Set a finalizer to be run. The Streaming Processor runs the SQL statement (a combination of insert, update, or delete statements, separated by semicolons) if no message is received from **sp_upload** within num milliseconds. The SQL statement is also run when **sp_upload** stops.

`-G`

Use Kerberos authentication. This option is required when the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller
2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`-h`
Print detailed help.

`-k` *`privateRsaKeyFile`*
Authentication is performed using the RSA private key file mechanism instead of password authentication. The *`privateRsaKeyFile`* must specify the pathname of the private RSA key file. This option is required if the Streaming Processor was started with the `-V rsa` option. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, the Streaming Processor must have been started with the `-k` option specifying the directory in which to store the RSA keys.

`-m` *`name`*
Set a symbolic tag name for the connection. This allows to look up the connection easily in the Aleri_Clients metadata stream. The name will be stored in the field “conn_tag”. To kill the connections by tag name, use the **sp_cli** command

`-p` *`[hostname:]port`*
Specify the port or the hostname and port of the Command and Control interface within a running instance of the Sybase Aleri Streaming Platform. If no hostname is specified, the default is "localhost".

`-r` *`num`*
Upload records or transaction blocks at a rate of num per second.

`-s` *`num`*
Synchronize the source streams every num records or transaction blocks. This guarantees that the records have been absorbed by the source streams.

`-t` *`num`*
Run in transaction mode. Each record that **sp_upload** reads is buffered on a per stream basis. When a buffer reaches the indicated number of records it is wrapped as a single transaction and sent to the Sybase Aleri Streaming Platform. If all records read are from one stream, this effectively buffers the stream into *`num`* records chunks and commits them as transactions. Any buffered records are sent as a single transaction per stream when an EOF is read.

`-w` *`num`*
Run in envelope mode. Each record that **sp_upload** reads is buffered on a per stream basis. When a buffer reaches the indicated number of records it is wrapped in a single envelope and sent to the Sybase Aleri Streaming Platform. If all records read are from one stream, this effectively buffers the stream into *`num`* records chunks. Any buffered records are sent as a single envelope per stream when an EOF is read.

`-x`
Upon receiving an EOF on the standard input, send an <END OF STREAM> marker to each stream for which data has been uploaded. If all source streams of the Sybase Aleri Streaming Platform receive an <END OF STREAM> marker, the Sybase Aleri Streaming Platform will shut down and exit.

Examples

To convert all XML records in file `foo.xml` to native binary format and post them to a running instance of the Sybase Aleri Streaming Platform:

```
cat foo.xml | sp_convert -p 11180 | sp_upload -c user:pass -p 11180
```

To convert all comma-separated records in the `foo.csv` file to native binary format and post them to a running instance of the Sybase Aleri Streaming Platform:

```
cat foo.csv | sp_convert -d "," -p 11180 | sp_upload -c user:pass -p 11180
```

To convert all XML records in the `foo.xml` file to native binary format and post them to a running instance of the Sybase Aleri Streaming Platform on a target machine HOST which has a differing byte order than the machine on which **sp_upload** is running:

```
cat foo.xml | sp_convert -b -p HOST:11180 | sp_upload -c user:pass -b -p HOST:11180
```

For a description of the format of the CSV and XML input files, refer to the sp_convert manpage.

Copyright

See Also

sp(1), sp_convert(1), sp_subscribe(1)

Bugs

See the documentation for known issues.

Name

sp_viewer — View the changing state of streams in the Sybase Aleri Streaming Platform.

Synopsis

`java` [-cp *`classpath`*] com.aleri.asap.tools.SPViewer [*`OPTION`*...]

Description

sp_viewer allows the user to subscribe to, and view, streams being maintained on a specified Sybase Aleri Streaming Platform instance. The program is written in Java, and uses a simple Swing GUI that allows the user to retrieve the stream list, subscribe to one or more streams, view one or more streams, and chart the update performance of a stream over time.

The Java class path must include the following jars:

- spviewer.jar
- commons-codec-1.1.jar
- xmlrpc-2.0.jar
- jfreechart-1.0.0-rc1.jar
- jcommon-1.0.0-rc1.jar
- bcprov-jdk14-134.jar

The jars can usually be specified as a colon-separated list in the Java command line option `-cp` or in the `CLASSPATH` environment variable. Full paths to the jars must be used.

Options

`c=`*`user`*[`:`*`passwd`*]

Pass authentication credentials to the Sybase Aleri Streaming Platform; if the password is not given, the user will be prompted for it. If the Sybase Aleri Streaming Platform successfully authenticates with these credentials, the connection is maintained, otherwise the Sybase Aleri Streaming Platform will immediately close the connection.

This option must correspond to the type of authentication specified for the Streaming Processor when it was started up. If it was started without specifying a `-V` option or using the `-V none` option, omit this option. If it was started using the `-V pam` option, specify `-c user:password`. If it was started using the `-V rsa` or `-V gssapi` option, specify `-c user`.

`e=on`|`off`

Turn encryption on or off; the default value is "off".

`G=on`|`off`

Use Kerberos authentication. This option is required when the the Streaming Processor was started with the `-V gssapi` option.

On a Windows server, if you have more than one valid Kerberos ticket, you must ensure that the ticket issued to the account used to connect to the Sybase Aleri Streaming Platform is the default ticket.

You may have obtained multiple valid Kerberos tickets if you:

1. authenticated using a Kerberos server other than your current domain controller

2. logged in to one user account and used a different one to connect to the Sybase Aleri Streaming Platform

`k=on|off`
Turn RSA authentication on or off; the default value is "off". If "on", the passwd argument in the `c=` option is the name of the private RSA key file.

`p=[`*`hostname`*`:]`*`port`*
Command and Control host name and port number. If the host is not specified, it defaults to "localhost". The port is required.

`t=`*`timerDelay`*
Specified in seconds, the timerDelay is used to acquire the current message count from the subscription thread at approximately every "timerDelay" interval, defaults to 10 seconds. The Java timer is not very accurate.

`x=`*`numSeconds`*
The X axis on the performance monitoring graph shows time, defaults to 0, enabling auto-range.

`y=`*`numUpdates`*
The Y axis of the performance monitoring graph shows the Sybase Aleri Streaming Platform subscription message count values, defaults to 0, enabling auto-range.

Usage Notes

Typically, the full Java command line, shown in the SYNOPSIS above, would be placed into a shell script that is then executed from the command line. Here is an example showing how to run the streamviewer application from the command line, where the jar files are located in the `../lib` directory, the Command and Control host and port are "ganges" and 11190 respectively, while the X axis (time series) is fixed at 180 seconds, the Y axis (platform message count) is fixed at a value of one hundred thousand, and the timerDelay interval (between message count sampling timer events) is 10 seconds:

```
SVCP=""
for i in \
  spviewer.jar commons-codec-1.1.jar xmlrpc-2.0.jar  \
  jfreechart-1.0.0-rc1.jar jcommon-1.0.0-rc1.jar bcprov-jdk14-134.jar
do
  SVCP=$SVCP:../lib/$i
done

java -cp "$SVCP" com.aleri.asap.tools.SPViewer \
  p=ganges:11190 c=userfoo:xyz123 e=off k=off x=180 y=100000 t=10
```

Relevant screens and menu items:

Main Screen
The main screen is used to retrieve the list of streams from the running platform instance. To retrieve the list of streams, drop down the "File" menu item, and select the "Retrieve Streams" menu item. From the main screen, the left mouse button (along with the **Ctrl** key) is used to highlight the stream(s) of interest. After selecting the streams, the right mouse button is used to pop up a control menu displaying a list of relevant options. The popup menu items follow next:

ViewStreamData
This menu item opens a screen that displays a tabular view of the

subscribed stream. From this screen you can also view the stream definition.

ViewStreamDataWithLog
This menu item opens a screen that displays a tabular view of the subscribed stream, along with a "log" status display, that shows the current update count, the state of the subscription thread, the type of the last update that came in, and the value of the key fields for the latest update.

ViewStreamPerformance
This screen is used to monitor the performance of a stream that has been subscribed to. The performance is displayed as a time series graph, where the X axis shows time, and the Y axis shows the Sybase Aleri Streaming Platform message count values. These values include the current update count, the total update count thus far, and the current running average of the update count. The default settings of the chart are acquired from the command line (see the `x`, `y`, and `t` command line options). The user has the option of changing the settings of the chart at run time using the "Settings" tab. In addition, the user can select the "auto-range" options in order to have the charting utility determine the settings dynamically as data flows in from the Sybase Aleri Streaming Platform instance.

Subscribe
This menu option is used to subscribe to the selected stream(s). Once this menu selection is made, a separate thread is spawned for each of the selected stream(s), and a subscription request is sent to the Gateway interface. This will start the update process flowing.

Unsubscribe
This menu option is used to unsubscribe from the selected streams.

DumpMsgCounts
This menu option is used to write the message counts of the subscription threads out to the Java console/standard out.

Notes

When monitoring a stream for performance, the timeDelay value is used to configure the interval, at which the Java timer event is fired. Each time the timer event is fired, the current platform message count will be acquired from the subscription thread that is absorbing data for the corresponding stream. The granularity of the timeDelay is in seconds. Unfortunately, the Java timer is not very accurate, and it often lags behind the expected firing time. In the future, the use of a more precise mechanism should be explored. In addition, the application should provide more robust charting options.

Copyright

Copyright 2010 Sybase, Inc. All Rights Reserved.

See Also

sp(1), sp_convert(1), sp_subscribe(1), sp_upload(1)

Bugs

See the documentation and release notes for known issues.

Chapter 6. Authoring Executables

Name

sp_encmodel — encrypt a Sybase data model

Synopsis

```
sp_encmodel [OPTION...] filename
```

Description

The **sp_encmodel** command is used to encrypt Sybase Aleri Streaming Platform data models to hide the intellectual property in commercial models from the end users. It reads in a data model file and writes out an encrypted data model file. There is no equivalent tool to read in an encrypted data model file and write out a decrypted data model file. The Sybase Aleri Streaming Platform decrypts the encrypted model file for parsing. Hence, the encrypted data model produced by **sp_encmodel** may be used wherever an unencrypted data model may be used: in arguments to sp_server(1), sp_convert(1), and other commands.

The suffix of the encrypted model's filename has no special meaning; it can be anything, including `.xml`. The programs recognize the encrypted models by the file contents. The metadata stream `Aleri_Config` will show the encrypted model. The sp_cli(1) commands that save the current model will also save it in the encrypted format.

The encrypted models can not be loaded into the Aleri Studio.

Required Arguments

filename Enter the name of the data model file that you wish to encrypt.

Options

`-` Take the input data model from standard in. This provides the ability to use **sp_encmodel** in a UNIX pipeline. For example, if you could have a single data model that required slightly different optimizations for different operating environments. With this option you can pipe the data model through a program to modify the model as required and then through **sp_encmodel** to encrypt it.

`-h` Enter this option to display the usage message for this command.

Examples

To read in a data model file named `model.xml` and write out an encrypted data model file named `model.enc`,

```
sp_encmodel model.xml > model.enc
```

To read in a data model from standard input (that has first been optimized for the Solaris operating system) and write out an encrypted data model file named `encryptedmodel.xml`,

```
cat model.xml | optimize -o solaris | sp_encmodel - > encryptedmodel.xml
```

Copyright

Copyright 2010 Sybase, Inc. All Rights Reserved.

See Also

sp_server(1)

Name

sp_sql2xml — Compile SQL statements and generate an Sybase Aleri Streaming Platform XML configuration file.

Synopsis

```
sp_sql2xml [OPTIONS]
```

Description

sp_sql2xml translates a given set of SQL statements to the corresponding XML representation in order to be consumed by the Sybase Aleri Streaming Platform. In addition, the translator verifies the correctness of the SQL statements, checks for data type consistencies and does limited optimization. For more information on the SQL syntax supported refer to the Service Authoring Language Syntax document.

Required Arguments

`-i` *`file`* use the specified file as the input to be translated to XML.

Options

`-o` *`file`* Write the XML output to the named file; this will overwrite the file if it exists. The XML is written to standard output by default.

`-s` *`file`* Use the file as the location of the XML Schema (`.xsd`) file used to validate the XML when submitted to the Sybase Aleri Streaming Platform. No check is done at translation time to verify the existence of the file; the file must exist at runtime.

Usage Notes

- All object names are case insensitive.
- Object names must begin with a letter or an underscore and can contain any number of letters, digits, and underscores.
- Each SQL statement can optionally be separated by a semi-colon.
- The translator stops at the first syntax error that it encounters and displays the line number and location of the error.
- Currently the same table cannot be used more than once in a given Join. If this functionality is required, create a copy of the table and then use the copy in the join.
- Every table in the join clause can have an alias. If an alias is not supplied, then the alias defaults to the table name. It is recommended that a two or three character alias be used for every table because when referring to columns in rules it must be prefixed with the table alias.
- When referencing a column in the current table being defined, in a rule, the column must not be prefixed with an alias.

Limits on Verification

Currently some verifications are not performed and as a result, if encountered, the XML produced will produce errors or unexpected results. The following verifications are not performed:

- There are no checks to ensure that all non-key columns in an aggregation have aggregation function. If such a condition exists then the Sybase Aleri Streaming Platform will catch the error and will not process the XML model.
- There are no checks to catch nested aggregation when writing a rule that refers to another column of the same table for which the rule is being written.
- Quoted field names will parse correctly but will produce invalid XML so avoid using them.

Unimplemented SQL Features

The following SQL features are not implemented currently.

- Calls to External Library Functions.
- Self Joins.
- The HAVING clause.
- Unions of Select Statements other than a simple SELECT * FROM <tablename>.
- CREATE SERVICE statements.
- The RETENTION property of a BaseStream.
- Any of the Alter Statements and Service related statements.

Copyright

See Also

sp(1)

Bugs

See the documentation for known issues.

Name

sp_studio — Script for starting the Sybase Visual Authoring Environment.

Synopsis

```
sp_studio
```

Description

sp_studio is a shell script that brings up the Aleri Studio. The environment can be used to visually author data models (in other words, continuous queries) and to kick off and monitor Streaming Processor execution of those services. Refer to the *Authoring Guide* for more information.

Copyright

See Also

sp(1)

Bugs

See the documentation for known issues.

Chapter 7. Advanced Debugging

This chapter introduces the command-line tools that can be used to debug and test a data model outside the Aleri Studio.

7.1. Introduction to Sybase Aleri Streaming Platform Debugging Tools

The bugs that crop up while creating a solid and efficient data model may not be easy to locate and fix. The Sybase Aleri Streaming Platform has a number of debugging features that can help with this task. These debugging features are available through two interfaces: the GUI in the Aleri Studio, and the command-line **sp_cli** tool.

The debugging tools are designed to be used during data model development; not while the Sybase Aleri Streaming Platform is in production mode. Debugging support places a substantial overhead on the Streaming Processor. That is why debugging features are normally disabled. To enable the debugging features, the system must be put into “trace” mode, which is different from the logging level referred to as debug level.

In trace mode, the Streaming Processor works as usual, but also performs extra checks for possible debugging operations and breakpoints and collects more information about the history of execution. The combination of extra synchronization overhead and extra checks can create more overhead.

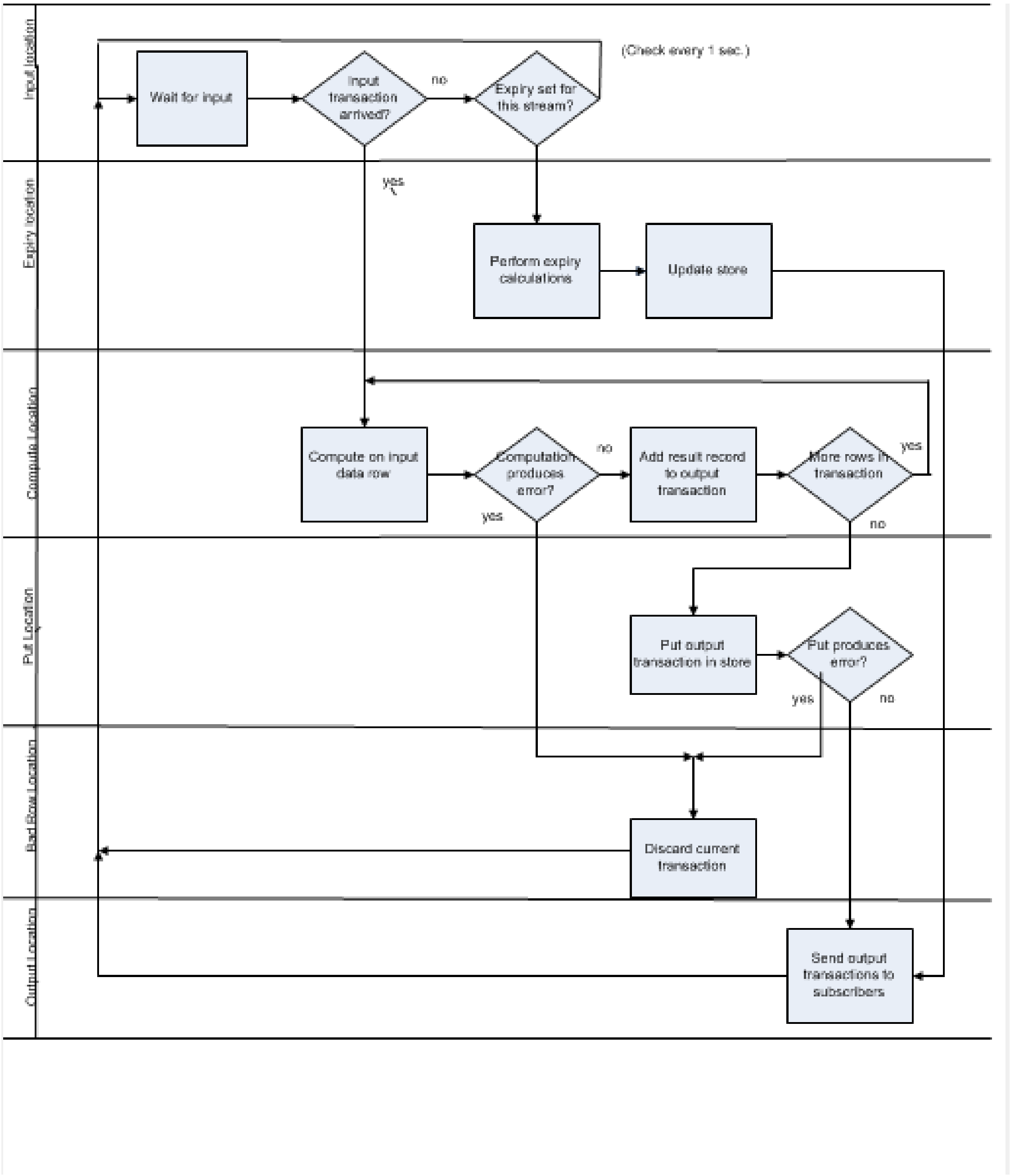
Trace mode is not supported on Streaming Processor instance running in a clustered configuration. Also not supported is using multiple copies of the Aleri Studio debugger on the same running instance of the Streaming Processor at the same time. And while the trace mode is enabled, the dynamic modifications can not be applied.

7.1.1. The Stream Processing Loop

To understand how the debugging tools work while the Streaming Processor is running in trace mode, it is helpful to explain how a stream on the Streaming Processor processes data.

7.1.1.1. “Locations” in the Stream Processing Loop

Each stream in the Sybase Aleri Streaming Platform is executed on a separate thread. The internal logic of a stream can be represented as an (almost) endless loop with the following state diagram:


Input location
Expiry location
Compute Location
Put Location
Bad Row Location
Output Location
(Check every 1 sec.)
Wait for input
Input transaction arrived?
no
yes
Expiry set for this stream?
Perform expiry calculations
Update store
Compute on input data row
Computation produces error?
no
yes
Add result record to output transaction
More rows in transaction
yes
no
Put output transaction in store
Put produces error?
yes
no
Discard current transaction
Send output transactions to subscribers

When talking about stream processing, these states are called “locations”.

A stream may be instructed manually to do a single step, to observe the processing of data in “freeze-frame motion”. A single step lets the stream proceed with the processing in the current location, and move to the next location. If a stream is waiting for I/O, it can't do anything, and any attempts to step it would just return immediately.

The normal processing sequence goes like this:

- INPUT:

  The stream waits for the input queue to become non-empty and then picks a transaction from the head of the input queue. This transaction becomes visible as `inTrans`, the current input transaction. It will be processed row-by-row.

The current output transaction is set to be empty, prepared to collect the results of processing. It becomes visible.

Then the stream enters the COMPUTE loop.

- COMPUTE:

  The next record is selected from the current input transaction. It becomes visible as `inRow`, the current input row. In some cases, the current input record may actually be two records, combined into an UPDATE_BLOCK (see the explanation below, in the PUT location).

  If this is not the first iteration of the loop, the records produced from processing the previous input record are still visible as `outRow`.

  If there are any input breakpoints (discussed below) defined on this stream, they are evaluated against the current input record, which may trigger a Sybase Aleri Streaming Platform pause.

  The check whether the Sybase Aleri Streaming Platform is paused (maybe by a breakpoint that has been triggered in this same stream on the previous step) is done. If paused, the stream pauses here and waits for permission to continue.

  Finally, the actual computation is done on the current input record. It produces zero or more output records. These records become visible as `outRow`, and are also appended to the end of `outTrans`. These records follow certain internal rules, and are not exactly the same as when they are published externally. For example, the update records at this point usually have the operation type UPSERT, and the delete records are SAFEDELETE.

  If there are any output breakpoints (discussed below) defined on this stream, they are evaluated against the current input record, which may trigger a Sybase Aleri Streaming Platform pause.

  If there are more records left in the input transaction, the compute loop continues; otherwise, the stream proceeds to put the calculated data into the store. That is, unless an exception such as division by zero has happened, in which case it proceeds to the BAD_ROW processing.

- PUT:

  The Sybase Aleri Streaming Platform is checked to see whether it's paused. If so, the stream pauses here and waits for permission to continue.

  The new result is placed into the stream's store. It's not a simple process, as the result transaction gets cleaned and transformed according to the information already in the store. Because of this, the current output transaction is no longer visible after this point. Of course, there is no current output row either. Some of these transformations are:

  - SAFEDELETEs are either thrown away (if there was no such record in the store) or converted to DELETEs (filled with all the data that they had in the store before being deleted).

  - UPSERTS are transformed into either INSERTS or UPDATE_BLOCKs. If any UPDATEs got here, they are converted to UPDATE_BLOCKs too, or may be simply discarded if no data is changed in the record from its previous state. An UPDATE_BLOCK is a pair of records. The first one has the operation type UPDATE_BLOCK and contains the new values, the second one has the operation type DELETE and contains the old values. When an UPDATE_BLOCK is published to outside the Sybase Aleri Streaming Platform, the second record is discarded and the first one is converted to an UPDATE. But here, inside the Sybase Aleri Streaming Platform, the whole update blocks are visible.

The PUT may trigger an exception too, for example, when trying to insert a record with a key that is already in the store. In this case the whole transaction is aborted and the stream moves to the BAD_ROW location.

If all went well, the current input transaction and current output transaction (already transformed) are inserted into the stream's history. The input transaction is added to the tail of `inHist`, the output transaction is added to the tail of `outHist`. Since the processing is done now, `inTrans` and `inRow` become invisible now. `outTrans` and `outRow` are already invisible by this time.

Then the stream moves to the OUTPUT location.

- OUTPUT:

  The result transaction is enqueued for publishing to the clients. If some clients are too slow and the output buffer fills, the stream would wait here for some buffer space to become free.

  Then the result transaction is delivered to any streams which have this stream as their input. Again, if any of their input queues become full, this stream would wait for them to become available.

  After that the stream goes to INPUT the next transaction.

Besides this main loop, there are a couple of side branches. For the streams with expiry, the following side branch occurs every second:

- EXPIRY:

  Prepare the expiry update. It becomes visible as `outTrans`.

  Then proceed just as with PUT (check pause, then put data to the store and output it).

In case of some errors in the data, the stream will enter the BAD_ROW location:

- BAD ROW:

  Make the rows with detected issues visible as `badRows`. The short messages describing what is wrong with them are visible as `badRowsReason`. There is one message per row. The same rows are written into the bad records file and the messages are reported on the Sybase Aleri Streaming Platform log.

  At this point `inTrans` still contains the transaction that has triggered the issue.

  Trigger an exception: ask the Sybase Aleri Streaming Platform to pause.

  Wait for permission to continue. Continue with INPUT of the next transaction.

7.1.1.2. Pausing the Streaming Processor in Trace Mode

While the Streaming Processor is running in trace mode, you can issue a command to pause this processing loop. This is where the distinction between processing and I/O locations becomes meaningful. While the Streaming Processor is in trace mode, the stream processing mechanism checks for a pause request as it enters each of the processing group locations. If one has been issued, the stream pauses and doesn't resume the loop until it is allowed to continue.

If the stream is engaged in actual processing when the pause is requested, the processing continues until entering the next processing location. Normally, it takes a very short amount of time, but if the stream is a badly programmed FlexStream that has just entered an endless loop, the pause would last indefinitely, or until the Streaming Processor shuts down.

The streams are not affected by a pause request when in an I/O location. The only type of “pause” that can happen in an I/O location is caused by the buffer each depends on. The stream will pause in the INPUT location if there are no transactions in the input queue. The stream will pause in the OUTPUT location only if the output queue is full.

The stream may move between the I/O locations to a processing location even when it is paused. IF there is a slow subscriber on a stream, the stream's output buffer would fill up, and the stream would sit in the OUTPUT location, waiting until the buffer space becomes available. If the stream in this location receives a pause request, the request is ignored. If the stream's subscriber takes a transaction off the output buffer after this request, the stream would deposit its current output transaction on the buffer and go to the INPUT location. At this point, after entering the INPUT location, the stream recognizes the pause request. No new data is processed after the pause request, but the stream does change location.

7.1.2. Trace Mode Basics

To enable trace mode with **sp_cli**, use:

```
sp_cli> trace_mode on
```

To disable it, use:

```
sp_cli> trace_mode off
```

To check whether the Sybase Aleri Streaming Platform is in trace mode, use:

```
sp_cli> trace_mode
```

For example:

```
sp_cli> trace_mode
trace mode is off
sp_cli> trace_mode on
sp_cli> trace_mode
trace mode is on
```

Trace mode is not connected to the instance of **sp_cli** — it's a mode of the Sybase Aleri Streaming Platform itself. You can start **sp_cli** and enable trace mode, and exit **sp_cli**. The Sybase Aleri Streaming Platform will stay in trace mode until it gets explicitly turned off.

Almost all of the debugging commands, with rare exceptions, work only in trace mode. Calling them with trace mode off is considered an error.

When the trace mode is turned off, most of the debugging state is preserved.

For example, the breakpoints will still be remembered, even though they won't be checked and triggered. Turning trace mode back on will reactivate breakpoints. But there are some exceptions, including the loss of the execution history since it's not updated with trace mode off.

7.2. Debugging in Trace Mode

This section discusses how to use the Streaming Processor debugging features in Trace mode.

In general, there are four main debugging activities:

- Pausing the Streaming Processor while a data model is running.
- Examining trace mode statistics for each stream at the pause point.
- Examining the streams' data stores and processing histories at the pause point.
- Stepping through the data model's processing cycle to the next pause point, and seeing what happens to the debugging statistics and processed data.

One group of Sybase Aleri Streaming Platform debugging tools enables you to pause and step the Streaming Processor manually or set breakpoints to automatically generate pauses. Another set allows you to view debugging data and actual streaming data.

7.2.1. Pausing the Streaming Processor

The Streaming Processor can be paused automatically by hitting a breakpoint or manually. To pause the Sybase Aleri Streaming Platform, it must be running in trace mode. You can use the **sp_cli** tool to pause the Streaming Processor, or check to see whether it is already paused, as follows:

```
sp_cli> check_pause
Platform is not paused
sp_cli> pause
sp_cli> check_pause
PAUSED
```

check_pause returns the Streaming Processor's current pause state.

If the Sybase Aleri Streaming Platform is already paused, issuing another **pause** command has no effect.

When the Sybase Aleri Streaming Platform is paused, no calculations happen inside it (although input and output may continue in the I/O locations and buffers). Any transactions in the streams' output buffers will still be consumable by the subscribers. Publishing to the paused Streaming Processor can continue until the streams' input buffers fill up.

To “unpause” and continue execution, use the following **sp_cli** command:

```
sp_cli> run
sp_cli> check_pause
Platform is not paused
```

If the Sybase Aleri Streaming Platform is already running, issuing another **run** command has no effect.

There is also a way to start the Sybase Aleri Streaming Platform and immediately pause it. This is convenient if the Streaming Processor's inputs can't be easily controlled, and start publishing data as soon as they connect to the Streaming Processor. If you start the Streaming Processor with this option, you will be able to single-step through all the data right from the beginning.

To do this, start **sp-opt** (or **sp**) with the `-DD` option:

```
sp-opt -DD ...
```

Another **sp_cli** command allows you to wait until the Sybase Aleri Streaming Platform is paused, by a breakpoint, another **sp_cli** command or the Aleri Studio:

```
sp_cli> wait_pause
```

This command is not interruptible (other than by killing **sp_cli**). It is useful in automated scripts that invoke **sp_cli**.

The current locations of the stream in a stopped Sybase Aleri Streaming Platform can be seen using the **spl_cli** command “ex” (for “examine”). The examples that follow show how this command is used.

7.2.1.1. Caveats when the Streaming Processor is Paused

When the Streaming Processor is paused, all the processing is frozen, which means that care should be taken with certain operations.

As mentioned, the paused Streaming Processor will continue to receive data published to it until the input queues fill up. At that point, the publishing will get stuck. A reliable publishing application will stop immediately when it fails to get any confirmation, but a less reliable one might continue sending data that the Streaming Processor does not receive.

To avoid this danger, it is best that publishers publish asynchronously, by a separate process or thread.

The same applies to reliable subscribers as well; they should send confirmations back to the Streaming Processor. Even when the Streaming Processor is paused, it's still possible to subscribe to the streams or get their current state.

The metadata streams are paused just like all other streams, and no updated should come from these streams while the Sybase Aleri Streaming Platform is paused. The sole exception is the special pseudo-stream Aleri_RunUpdates.

Dynamic modifications cannot be done when the Sybase Aleri Streaming Platform is in trace mode, paused or not. Tracing clustered configurations are also not supported.

7.2.2. A Simple Example

```
<?xml version="1.0" encoding="UTF-8"?>

<Platform version="3.2">
    <Store file="store" fullsize="20" id="store"/>

    <SourceStream id="filterInput" ofile="output/filterInput.out"
store="store" type="dynamic">
    <Column datatype="int32" key="true" name="a"/>
    <Column datatype="string" key="false" name="b"/>
    <Column datatype="int32" key="false" name="intData"/>
  </SourceStream>
    <FilterStream id="filter" istream="filterInput"
ofile="output/filter.out" store="store">
        <FilterExpression>(10 / filterInput.intData) &lt;
1</FilterExpression>
    </FilterStream>

</Platform>
```

With the following input, fed by two records per transaction:

```
<filterInput ALERI_OPS="i" a="1" b="a" intData="10" />
<filterInput ALERI_OPS="i" a="2" b="a" intData="20" />

<filterInput ALERI_OPS="i" a="3" b="a" intData="20" />
<filterInput ALERI_OPS="u" a="4" b="a" intData="10" />

<filterInput ALERI_OPS="u" a="2" b="a" intData="10" />
<filterInput ALERI_OPS="u" a="1" b="a" intData="20" />

<filterInput ALERI_OPS="d" a="2" />
```

The state of the streams in this model may be examined as:

```
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="1" outSeq="0" stepSeq="0"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="0" inSeq="0" outSeq="0" stepSeq="0"/>
```

The **pause** argument tells the Streaming Processor to display the state of all paused streams. The curly braces around **pause** are a newer way of quoting in **sp_cli** (supported since Release 2.4.) but the other type with back quotes would also work. One type must be chosen. If you just enter what is below:

```
sp_cli> ex pause
```

you would get a syntax error, because `pause` is also a keyword and the name of another command.

The state data is displayed in an XML format, one record per stream. The stream `filterInput` is in the COMPUTE location, about to do calculation. The stream `filter` is waiting for input.

The other fields provide extra information about the state of streams.

None of the streams has any breakpoints triggered. The field `onbp` being equal to 0 indicates this. If any breakpoints had been triggered, the value of `onbp` would be the ID of that breakpoint.

`Throttle` shows the stream's input buffer size. When the input queue grows over this size, the stream stops accepting further input. The streams use double-buffering for the queues, so the point where the input would stop being accepted would actually be somewhere between the throttle size and twice the throttle size.

`History` shows the size of the history kept for the stream. As the stream runs in trace mode, it remembers the pairs of input transactions received and output transactions produced from them. The history size determines how many of the most recent pairs will be kept. When a breakpoint is reached, the last transaction might be just the “tip of the iceberg” since the really interesting events could have happened earlier. The history gives you the ability to see previous transactions.

The sequence values, except `stepSeq`, tick whenever a transaction is moved around:

postSeq when a transaction is placed onto the stream's input queue

inSeq whenever a transaction is read off the input queue for processing outSeq whenever a transaction is sent to the output (and added to the history, together with the matching input record)

The difference between `postSeq` and `inSeq` gives you the current size of the input queue. The input and output sequences mostly go together but there are obscure cases that add discrepancies. For example, an expiry action produces output transactions without any input transactions. Internal service transactions are counted on the input but not on the output, and are not recorded in the history. These sequences tick even when trace mode is off.

In this example, the stream `filterInput` is processing the first transaction, about to compute a result for one of the input rows. It has three more transactions queued up. It has not produced any output yet, so the stream `filter` is still waiting for any data to come in.

The value of `stepSeq` only gets increased in trace mode, whenever a stream does a processing "step", either during normal running or in a single-stepping mode. Usually the processing of a transaction involves at least two steps.

The **ex {pause}** command displays data only for user-defined streams. The Streaming Processor also contains a number of metadata streams. These streams are subject to pausing, just like other streams. To see the state of all streams use the **ex {pauseAll}** command.

7.2.3. Example Part 2: Single-Stepping

Continuing the previous example, tell the `filterInput` stream to do a single step:

```
sp_cli> step {filterInput}
```

You can also single-step the stream `filter`, but since it's in the INPUT location, it won't be useful. This call would have no effect.

In some cases, the Sybase Aleri Streaming Platform will move to the next freeze-frame. To do this, just step any stream, and let the Sybase Aleri Streaming Platform pick a stream that has some processing to do. This can be done with:

```
sp_cli> step
```

As the stream moves through locations, you can examine information about its data. Some data, like the contents of the input queue, can be examined at any time. Other data elements are visible only at certain times. For example, the current input transaction can be seen only when the stream is processing a transaction. These data elements can be examined at other time, but the values would be zero.

The example above ended just before the stream `filterInput` was to process the first row. Below is the first transaction and what row is about to be processed:

```
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {inRow} {filterInput}
  <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
```

(In these printed displays, the indentation has been massaged a little to make the records more readable as they wrap to the second line).

The XML element tag in the printout is the stream's name, not just <row>. For the data elements related to the input data, the tag is the name of the stream that produced the data. This is important since a stream may get input from multiple sources. In this case, `filterInput` is a source stream, so its own name is used in the printout.

Now process the first row of the transaction.

```
sp_cli> step {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="1" outSeq="0" stepSeq="1"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
      history="100" postSeq="0" inSeq="0" outSeq="0" stepSeq="0"/>
```

The value in the `stepSeq` field of the `filterInput` has increased, showing that this stream has made a step. It is still in the COMPUTE location but about to process the second row:

```
sp_cli> ex {inRow} {filterInput}
  <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
```

The result of the processing of the first row:

```
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
sp_cli> ex {outRow} {filterInput}
  <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
```

At this point, the output transaction contains only this one row. Another step is needed:

```
sp_cli> step {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="1" outSeq="0" stepSeq="2"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="0" inSeq="0" outSeq="0" stepSeq="0"/>
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
  <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
sp_cli> ex {outRow} {filterInput}
    <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
```

One more row has been added to the output transaction. This one is not wrapped in <trans> tags. That's because the output transaction is still being built. Since `outTrans` may not contain more than rows for one transaction, there is no need at this point to track the transaction boundaries inside it.

The PUT location shows now the stream is done with processing the transaction, and is ready to put it into the store. Proceed to the next step:

```
sp_cli> step {filterInput}
sp_cli> ex {pause}
```

```
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="3"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
sp_cli> ex {outTrans} {filterInput}
```

The `filter` stream is now also in the COMPUTE location: it has found the data posted to it by `filterInput` and is prepared to process it. But it is paused at that location until it is allowed to proceed.

The `filterInput` stream is ready to process the next transaction. `outTrans` is now empty, as it has been reset. The previous input and output transaction can be found in the history:

```
sp_cli> ex {inHist} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {outHist} {filterInput}
  <trans>
    <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
```

Now the output transaction is fully formed. The history for this stream shows us the output transactions in mixed form (in matched input/output pairs):

```
sp_cli> ex {hist} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
  <trans>
    <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
```

In this case, the history contains only one transaction, so it's easy to get to the last transaction. But you can also tell **sp_cli** to show only the last transaction.

```
sp_cli> ex {lastInTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {lastOutTrans} {filterInput}
  <trans>
    <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {lastTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
  <trans>
    <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
```

Continuing with the next transaction:

```
sp_cli> ex {inRow} {filterInput}
  <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
sp_cli> step {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="4"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {outRow} {filterInput}
  <row ALERI_OPS="i"  a="3" b="a" intData="20"/>
```

One more row got processed. In the pending row, there is something unusual:

```
sp_cli> ex {inRow} {filterInput}
  <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
```

The problem is the stream is trying to update the record with key `a=4`, but this key hasn't been inserted into the stream yet. Continuing on:

```
sp_cli> step {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="5"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="i"  a="3" b="a" intData="20"/>
  <row ALERI_OPS="u"  a="4" b="a" intData="10"/>
```

In the COMPUTE location, the stream just processes the record without knowing whether it can be inserted or not. The error surfaces on an attempt to put the records into the store:

```
sp_cli> step {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="BAD_ROW" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="6"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
sp_cli> ex {outTrans} {filterInput}
sp_cli> ex {badRows} {filterInput}
  <row ALERI_OPS="u"  a="4" b="a" intData="10"/>
sp_cli> ex {badRowsReason} {filterInput}
  <row ALERI_OPS="i"  reason="Bad update writing to store."/>
```

The stream filterInput is in the BAD_ROW location. The input transaction is still showing fine but the output transaction is cleared (since the transactions with errors are discarded). By examining badRows and badRowsReason you can view the offending rows, and find out what went wrong with them.

Doing the next step will clear the error and continue computations:

```
sp_cli> step {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="3" outSeq="2" stepSeq="7"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {hist} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
  <trans>
    <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
  <row ALERI_OPS="i" />
```

The history now contains two pairs of transactions. It shows that this transaction resulted in no output. There are two interesting things about the last output transaction in this history:

- The output record contains no fields at all, not even the key fields, and the operation is “N” (for NOP, or “no operation”). This is a special placeholder record, used to show that no actual data went out.
- The output record is not enclosed in <trans>. When a transaction contains one record only, it's shown as just this one record, without the <trans> node.

The `postSeq` for the `filter` stream was not increased this time. Since the bad transaction was thrown away and produced no output, there was nothing to post to the input queue of `filter`.

7.2.4. Changing the History Size Limit

The history size limit can be changed, for one stream or for all streams. Continuing with this example, but changing the history limit of `filterInput` to 1:

```
sp_cli> history {1} {filterInput}
sp_cli> history ex {filterInput}
  1
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="1" postSeq="4" inSeq="3" outSeq="2" stepSeq="7"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {hist} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
  <row ALERI_OPS="N" />
```

The **history ex** command is an alternative way to see the history size limit. It works even with trace mode off. The history limit has changed for `filterInput` but stayed the same for `filter`. As the limit was reduced, the history of `filterInput` was truncated to only one pair of transactions.

Now change all the history limits to 10:

```
sp_cli> history {10}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="10" postSeq="4" inSeq="3" outSeq="2" stepSeq="7"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="10" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {hist} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
  <row ALERI_OPS="N" />
```

Both streams had their history value changed. The history of `filterInput` still contains only one pair of transactions so increasing the limit didn't bring the contents back. But the new contents may now accumulate, up to 10 pairs.

7.3. Automatic single-stepping

Manual single-stepping gets tedious quickly. For this reason, the Streaming Processor provides automatic stepping commands. They are functionally similar to the "step-over-a-call" commands in other program debuggers.

Automatic stepping does not work with breakpoints or bad row exceptions. If any of these occur, they are reported to the Aleri Studio but do not stop the stepping. The reason is that automatic stepping is intended to let the not-so-interesting data pass through the Streaming Processor conveniently, independently of breakpoints.

The first auto-step command, **step trans**, is used to step to the end of transaction. It does at least one common step, and then continues stepping as long as the stream stays in the COMPUTE location. This means that it stops when the stream moves into the PUT or BAD_ROW location and allows you to examine the transaction's effect before committing or discarding it. The first unconditional step is to get from this position over a pending PUT or BAD_ROW to the computation of the next transaction. You can call **step trans** repeatedly to step past transactions.

If the execution blocks in the INPUT or OUTPUT location for longer than 0.3 second, **step trans** stops.

Continuing the previous example:

```
sp_cli> step trans {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="0" throttle="512"
    history="10" postSeq="4" inSeq="3" outSeq="2" stepSeq="9"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="0" throttle="512"
    history="10" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="u"  a="2" b="a" intData="10"/>
    <filterInput ALERI_OPS="u"  a="1" b="a" intData="20"/>
  </trans>
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="u"  a="2" b="a" intData="10"/>
  <row ALERI_OPS="u"  a="1" b="a" intData="20"/>
sp_cli> ex {hist} {filterInput}
  <trans>
```

```
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
  <row ALERI_OPS="N" />
```

The execution has stopped in the PUT location. The transaction has been fully computed, and ready to be put to the store. The history has not changed since the last time (because the transaction has not completed yet).

The rest of the auto-stepping commands are related to the concept of quiescence (running the streams until they have processed all the available input). These commands are:

step quiesce stream {streamName}
Automatically step the stream and all its direct and indirect descendants until all of them are quiesced (that is, until all their input queues are empty).

step quiesce downstream {streamName}
Similar to **step quiesce stream**, but only the stream's descendants are stepped: not the stream itself. This command is convenient to clear out the descendant streams' input queues. Then when the argument stream will produce its output, the progression of the data through the descendant streams can be easily traced.

step quiesce from base
Automatically step all the derived (non-source) streams until their input queues are empty. This command is useful to clean out the queues of derived streams before processing an interesting record through the source stream. Then the progression of data through the derived streams can be easily watched.

Unlike the **step trace** command, these commands don't stop at the PUT location. Instead, they run until all the concerned streams are blocked on INPUT or OUTPUT. A stream is not quiesced if it is blocked on OUTPUT. Since the Streaming Processor cannot do anything about that, it stops auto-stepping anyway, so that you can fix the cause (such as an external program not reading the data fast enough) and call the auto-stepping again.

Continue the example by quiescing downstream from `filterInput`. Since this model contains only one stream, `filter`, derived from `filterInput`, the command quiesces `filter`.

```
sp_cli> step quiesce downstream {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="0" throttle="512"
    history="10" postSeq="4" inSeq="3" outSeq="2" stepSeq="9"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="10" postSeq="1" inSeq="1" outSeq="1" stepSeq="3"/>
sp_cli> ex {hist} {filter}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
  <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
```

The `filter` stream is now quiesced and waiting for more input. It has processed the sole transaction it had pending. The output of that transaction is a single row that passed through the filter.

Running "step quiesce downstream {filterInput}" would have the same effect in this example as **step quiesce from base**, since `filter` is the only stream derived from `filterInput`.

Finally, process all the data left in the model by quiescing the stream `filterInput`:

```
sp_cli> step quiesce stream {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="INPUT" onbp="0" throttle="512"
    history="10" postSeq="4" inSeq="4" outSeq="4" stepSeq="12"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="10" postSeq="3" inSeq="3" outSeq="3" stepSeq="8"/>
```

All the data has been processed and both streams are waiting for more input. See what happened with the data:

```
sp_cli> ex {hist} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
  <row ALERI_OPS="N" />
  <trans>
    <filterInput ALERI_OPS="u"  a="2" b="a" intData="10"/>
    <filterInput ALERI_OPS="u"  a="1" b="a" intData="20"/>
  </trans>
  <trans>
    <pair>
      <row ALERI_OPS="u"  a="2" b="a" intData="10"/>
      <row ALERI_OPS="d"  a="2" b="a" intData="20"/>
    </pair>
    <pair>
      <row ALERI_OPS="u"  a="1" b="a" intData="20"/>
      <row ALERI_OPS="d"  a="1" b="a" intData="10"/>
    </pair>
  </trans>
  <filterInput ALERI_OPS="d"  a="2"/>
  <row ALERI_OPS="d"  a="2" b="a" intData="10"/>
```

The next transaction contained an update for the rows with keys "1" and "2". When these records were put into the store, they became transformed from UPDATEs to UPDATE_BLOCKs. An UPDATE_BLOCK is a pair of records: the first one contains the updated values, the second one contains the old values that are being replaced.

The last transaction consisted of a single record that deleted the record with the key "2". When it was put to the store, all the fields were filled in with the values from the record that is being deleted.

See how the data progressed through the filter:

```
sp_cli> ex {hist} {filter}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
  <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
  <trans>
    <pair>
      <filterInput ALERI_OPS="u"  a="2" b="a" intData="10"/>
      <filterInput ALERI_OPS="d"  a="2" b="a" intData="20"/>
    </pair>
    <pair>
      <filterInput ALERI_OPS="u"  a="1" b="a" intData="20"/>
      <filterInput ALERI_OPS="d"  a="1" b="a" intData="10"/>
    </pair>
  </trans>
  <trans>
    <row ALERI_OPS="d"  a="2" b="a" intData="20"/>
    <row ALERI_OPS="i"  a="1" b="a" intData="20"/>
  </trans>
  <filterInput ALERI_OPS="d"  a="2" b="a" intData="10"/>
  <row ALERI_OPS="N" />
```

The transaction with updates became a delete and an insert because the updates have changed the filtering of the rows. The row with key "2" that was previously allowed to pass through the filter is not allowed to do so, so it's deleted from the output of the filter. In contrast, the row with key "1" now satisfies the filtering condition, so it's inserted to the filter output.

The final transaction that deletes the row with key "2" has no effect on the output of the filter. This row was previously blocked by the filter.

As the very final step in this example, stop the Sybase Aleri Streaming Platform:

```
sp_cli> stop
```

Note:

Be careful when using the debugger. If you exit **sp_cli** or the Aleri Studio while the Streaming Processor is in a paused state, it will stay paused. When you connect with another instance of **sp_cli**, it will still be paused. The same goes for trace mode: if you leave the Streaming Processor in trace mode, it will stay in that mode: if it encounters a breakpoint or exception, it will pause, and stop all processing until unpaused.

The Streaming Processor can be stopped from **sp_cli** even when it's paused. If you do this, the Streaming Processor will be unpaused, trace mode will be disabled, and then the Streaming Processor will proceed to stop and exit as usual.

Disabling trace mode will also unpause the Sybase Aleri Streaming Platform.

7.3.1. An extra parameter and its creative uses

All the auto-stepping commands have an additional (optional) argument: the value that limits the number of steps that can be taken automatically. This parameter can be used to put an upper bound on the time of execution of these commands, since they are not interruptible. Once the limit is reached, the stepping stops, and the debugging tool returns an appropriate error code. Debugging tools such as those in the Aleri Studio can then use this as a polling point — an opportunity to check on whether the user has decided to cancel the command.

In some cases, it can be an important capability. If the argument of the **step quiesce stream** command is a source stream that continues to receive and dispatch data, this type of limit would be the only way to stop the stepping.

sp_cli does not reissue these commands automatically. It just has applies some reasonable default limits. Of course, these limits can also be specified explicitly.

These limits also have some creative uses, such as the **step** command without an argument, that steps any stream in the Sybase Aleri Streaming Platform. It can be used as an auto-stepping command with a limit of 1 to do a similar thing but step only the descendants of some interesting stream:

```
sp_cli> step quiesce downstream {streamName} {1}
```

Or to step any derived stream at random, you can use:

```
sp_cli> step quiesce from base {1}
```

The limit can be changed (to 10, for example) to make things quicker. But then it will not be easy to tell afterward which streams were stepped and by how many steps.

7.4. Breakpoints and exceptions

The breakpoint mechanism provides a way to let the Streaming Processor run unattended, and then pause when something interesting occurs. This occurrence is usually related to a problem with the data model. When this happens, the breakpoint should stop the Streaming Processor so you can troubleshoot the problem.

The most basic kinds of problems are:

- A wrong record comes out of some stream.
- A record should come out as a result of another record but fails to do so or comes out in the wrong form.
- A stream detects a bad row.

You could detect these problems at the following places:

Wrong result record	At the output of a stream
Unexpected action to an input record	At the input of a stream, when the target record enters it
Bad record	When the stream reports it

A bad record is always a problem, so the Streaming Processor always pauses when one is encountered (this is when a stream enters a BAD_ROW location). This kind of response can be set up with exceptions, discussed further below. For other situations, the debugging tool provides user-defined breakpoints.

All breakpoints are set on a stream. There are two kinds of breakpoints:

- **“on input”**

  These breakpoints are checked when a row is taken from the input transaction for processing, before any computation is done on it. The Streaming Processor would be paused in the COMPUTE location.

- **“on output”**

  These breakpoints are checked after a row has been processed. The Streaming Processor would be paused in the next location (COMPUTE for the next input row, PUT or BAD_ROW).

This is called "the origin of a breakpoint". Since a stream may take input from multiple other streams, the "on input" breakpoints are further divided by origin:

- **“on any input”**

Checked for input from any stream.

- **“on a particular stream”**

  Checked only when the input transaction is coming from a particular stream.

7.4.1. Unconditional breakpoints and exceptions

In the simplest case, breakpoints are unconditional: once the stream is in the right location, the breakpoint gets triggered and the whole Streaming Processor is paused. More than one breakpoint may get triggered at the same time, since each stream runs in its own thread, so each one can trigger a breakpoint independently. There may be a simpler reason as well: nothing stops you from defining multiple breakpoints on the exact same condition. The Streaming Processor can tell one breakpoint from another by the unique ID assigned when the breakpoint gets created. If you create two breakpoints that are the same, the Streaming Processor gives each one a separate ID, and treats them as completely separate. So both could get triggered at the same time.

Once a breakpoint is created, it can be enabled or disabled, and slightly altered, but little more can be changed. The breakpoint can't be moved to another location or have its conditional expression changed. To make major changes, you must delete the breakpoint and create a new one.

The **sp_cli** commands for manipulating breakpoints start with `bp`. Restart the simple example with the same input, pause it, and add some breakpoints:

```
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="1" outSeq="0" stepSeq="0"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="0" inSeq="0" outSeq="0" stepSeq="0"/>
sp_cli> bp add {filterInput} any
  1
sp_cli> bp add {filterInput} out
  2
sp_cli> bp add {filter} {filterInput}
  3
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
sp_cli> ex {breakpoints}
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
```

The streams have been paused before processing the inputs and defining three breakpoints:

- On any input to `filterInput`.
- On output of `filterInput`.
- On the input of `filter`, when the data comes from `filterInput`. Since `filter` has only one input, there is really not much difference whether you specify the particular input stream or just any

input. There will be more differences, though, with conditional breakpoints.

Note that `any` and `out` are keywords: they must be written without quotes. The return from the command that creates a breakpoint is its assigned ID.

After the breakpoints are created, the command to print out is, **bp list**, which is a convenient synonym for **ex {breakpoints}**. The fields in the “printout” for a breakpoint are:

id — The ID of the breakpoint.

stream — The stream on which the breakpoint is defined.

origin — The location in the stream serving as the breakpoint's origin. It may contain the name of the input stream or "*" for any input stream, or it may be empty for a breakpoint on output.

expr — Conditional expression. This will be discussed later.

enabledEvery, leftToTrigger — Allow the breakpoint to trigger not on every occasion but on every Nth occasion. See more below.

onit — Set to 1 when the breakpoint is triggered, otherwise set to 0.

Sometimes it might be desirable to skip some number of records and then pause the Sybase Aleri Streaming Platform. For example, if some bug surfaces on the 1000th record passing though a stream, it would be convenient to let 999 records pass, and then pause and single-step from there. To allow this, configure a breakpoint to be triggered on every Nth row. If a breakpoint is put on input of this stream and configure it to be triggered on every 1000th row, then it would pass 999 records through and break on row 1000. If the Streaming Processor is restarted, the breakpoint would trigger next time on the 2000th row. In `bp list` the field `enabledEvery` shows this number N, and `leftToTrigger` shows how many records remain to be seen before the breakpoint gets triggered. Every time the breakpoint is triggered, `leftToTrigger` is reset to the original value of `enabledEvery`. By default when a breakpoint is created, `enabledEvery` is set to 1, to trigger on each row. It can be changed with the **sp_cli** command **bp every**.

Another convenience is the ability to disable a breakpoint temporarily. That can be done by configuring the breakpoint to trigger on every 0th record. The command **bp on** is a shorter way to do this. **bp on** restores the breakpoint back to be triggered on each record. For example:

```
sp_cli> bp off all
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="0" leftToTrigger="0" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="0" leftToTrigger="0" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="0" leftToTrigger="0" onit="0"/>
sp_cli> bp every {100} {3}
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="0" leftToTrigger="0" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="0" leftToTrigger="0" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="100" leftToTrigger="100" onit="0"/>
sp_cli> bp on {1}
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="0" leftToTrigger="0" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="100" leftToTrigger="100" onit="0"/>
sp_cli> bp every {2} all
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="2" leftToTrigger="2" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="2" leftToTrigger="2" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="2" leftToTrigger="2" onit="0"/>
sp_cli> bp on {1}
sp_cli> bp every {1} {3}
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="2" leftToTrigger="2" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
```

All these commands take either the ID of a breakpoint or the keyword "all" as the last argument. Breakpoint 2 has been configured (on the output of `filterInput`) to be triggered on every 2 records, and the other two breakpoints on each record. Let the Streaming Processor run and see where it stops:

```
sp_cli> run
sp_cli> wait_pause
```

In practice, the first breakpoint would get triggered before you can type **wait_pause**, but it's included here for the sake of completeness.

```
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="COMPUTE" onbp="1" throttle="512"
    history="100" postSeq="4" inSeq="1" outSeq="0" stepSeq="1"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="0" inSeq="0" outSeq="0" stepSeq="0"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="1"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="2" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
```

The **ex {pause}** command shows that `filterInput` has been stopped on the breakpoint 1 ("onbp"). And the **bp list** command also shows that breakpoint 1 has been triggered ("onit"). Multiple breakpoints can be triggered on the same stream at the same time. The **ex {pause}** command would show the ID of only one of them, at random. But the **bp list** command would show all the triggered breakpoints.)

The triggered breakpoint is an input breakpoint, and `filterStream` is ready to do the computation for it.

```
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {inRow} {filterInput}
  <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
```

```
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
```

The second input row is about to be computed. Look closely at the output of **bp list**: it shows that breakpoint 2 has only one record left to trigger. This means that the execution went past it once.

Remember the point when the Streaming Processor was paused initially, `filterInput` had picked up the first transaction from the input and was about to compute it. When breakpoints were added, `filterStream` was already past the place where it would check for breakpoint 1. In fact, it was stopped in exactly the same location where it would get stopped on breakpoint 1. Since the breakpoints are not immediately checked when they are added, this went unnoticed. When the Streaming Processor restarted, the execution continued past that point; the first time that breakpoint 1 was checked was when `filterInput` prepared to compute the second record.

To continue:

```
sp_cli> run
sp_cli> wait_pause
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="2" throttle="512"
    history="100" postSeq="4" inSeq="1" outSeq="0" stepSeq="2"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="0" inSeq="0" outSeq="0" stepSeq="0"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="2" leftToTrigger="2" onit="1"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="i"  a="1" b="a" intData="10"/>
  <row ALERI_OPS="i"  a="2" b="a" intData="20"/>
```

In this part of the example, the second record got processed, and breakpoint 2 triggered. Since this was the last record of the transaction, the stream `filterInput` is now in the PUT location.

Use the commands from the previous section to auto-step through one transaction on `filterInput`:

```
sp_cli> step trans {filterInput}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="2" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="5"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="3" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="0" stepSeq="0"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="2" leftToTrigger="2" onit="1"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="1"/>
sp_cli> ex {inTrans} {filterInput}
  <trans>
    <filterInput ALERI_OPS="i"  a="3" b="a" intData="20"/>
    <filterInput ALERI_OPS="u"  a="4" b="a" intData="10"/>
  </trans>
```

```
sp_cli> ex {outTrans} {filterInput}
  <row ALERI_OPS="i"  a="3" b="a" intData="20"/>
  <row ALERI_OPS="u"  a="4" b="a" intData="10"/>
sp_cli> ex {inTrans} {filter}
  <trans>
    <filterInput ALERI_OPS="i"  a="1" b="a" intData="10"/>
    <filterInput ALERI_OPS="i"  a="2" b="a" intData="20"/>
  </trans>
sp_cli> ex {outTrans} {filter}
```

The stream filterInput is at the PUT location of the second transaction. The data shows that the breakpoints 2 and 3 were triggered. After the PUT and OUTPUT of the first transaction in filterInput completed, filter received the transaction in its input record, and prepared to process it. This triggered breakpoint 3 (which is on input from filterInput). By then filterInput had cycled twice through the COMPUTE location, and on the second pass breakpoint 2 got triggered.

The cycle through COMPUTE must have triggered breakpoint 1 twice as well. But the auto-stepping mode ignored the breakpoints: they didn't pause the execution. The stepping can only happen on an already paused Streaming Processor. The Streaming Processor doesn't get unpaused for stepping; instead, some special internal mechanics are used to let certain streams proceed to the next location.

If the Streaming Processor is running now, there are two streams that have something to do. Both are about to trigger another pause: filterInput is about to put an output transaction its store that would trigger a BAD_ROW exception. filter will process the current row and then trigger breakpoint 3 as it picks the next row of the current transaction to compute. Whether one or both are triggered (and if just one, which one) is up to the thread scheduler of your machine and operating system. The element of chance can be taken out of this situation by quiescing the filter stream first.

```
sp_cli> step quiesce stream {filter}
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="PUT" onbp="2" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="5"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="1" stepSeq="3"/>
sp_cli> run
sp_cli> wait_pause
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="BAD_ROW" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="2" outSeq="1" stepSeq="6"/>
  <row ALERI_OPS="i"  name="filter" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="1" inSeq="1" outSeq="1" stepSeq="3"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="1" stream="filterInput" origin="*" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="2" stream="filterInput" origin="" expr=""
    enabledEvery="2" leftToTrigger="2" onit="0"/>
  <row ALERI_OPS="i"  id="3" stream="filter" origin="filterInput" expr=""
    enabledEvery="1" leftToTrigger="1" onit="0"/>
```

None of the breakpoints were triggered but the Streaming Processor got paused. The location BAD_ROW of filterInput is the explanation. Whenever a stream enters the BAD_ROW location, it triggers an exception that works just like a breakpoint: it pauses the Sybase Aleri Streaming Platform. Now examine badRows and badRowsReason as in the previous section.

Before moving on to the conditional breakpoints, delete the unconditional breakpoints created:

```
sp_cli> bp del all
sp_cli> bp list
```

The list of breakpoints is now empty. You can delete an individual breakpoint by entering its ID instead of "all".

7.4.2. Conditional breakpoints

There are times when more than unconditional breakpoints are needed, such as if you want to see a record with certain contents pass through a stream.

It can be done on the Streaming Processor by specifying a filter expression for a breakpoint. The filter expression is evaluated first and if it results in a false (0 or NULL) value, the breakpoint is skipped. The breakpoint is triggered only if the expression evaluates to true (or its `leftToTrigger` count is reduced).

The filter expression is a normal SPLASH expression. It may use the data from two pre-defined record variables: `row` and `oldrow`. `row` contains the current record; `oldrow` is defined only for the breakpoints on input. “oldrow” would be NULL for INSERTs and plain UPDATEs. For the UPDATE_BLOCKs `oldrow` will contain the second record of the block, the old data that is being replaced. For DELETEs and SAFEDELETEs `oldrow` contains the same data as `row`. A particular field can be accessed using the usual `row.field` syntax, and you can get the row operation code using **getOpcode(row)**.

The Row Definition that provides these predefined variables changes with different types of breakpoints.

For a breakpoint on output it's the Row Definition of the stream where the breakpoint is defined. The expression is evaluated on the output rows produced during the preceding COMPUTE. Since multiple rows can be produced, the expression is evaluated on each of them. If no rows are produced, the expression is still evaluated once with `row` set to NULL. In this case, `oldrow` is not available, since the UPDATE_BLOCKs are never produced on output of COMPUTE.

For a breakpoint on input from a specific stream it's the Row Definition of that input stream. The expression is evaluated on the record or update block that is about to be computed.

For a breakpoint on any input there is an ambiguity. Filter expressions are not permitted for this kind of breakpoint.

A source stream gets data from outside the Sybase Aleri Streaming Platform instead of other streams, so to put a conditional breakpoint on the input of a source stream, you would use the source stream's own name for the input stream by **bp add {filterInput} {filterInput}**.

To avoid the uncertainty when adding breakpoints about which one gets triggered first, breakpoints should be added only to the stream `filter`.

```
sp_cli> bp add {filter} {filterInput} {row.a = 2}
  4
sp_cli> bp add {filter} out {isnull(row)}
  5
sp_cli> run
sp_cli> wait_pause
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="4" outSeq="4" stepSeq="12"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="4" throttle="512"
    history="100" postSeq="3" inSeq="2" outSeq="1" stepSeq="3"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="4" stream="filter" origin="filterInput"
    expr="row.a = 2" enabledEvery="1" leftToTrigger="1" onit="1"/>
  <row ALERI_OPS="i"  id="5" stream="filter" origin=""
    expr="isnull(row)" enabledEvery="1" leftToTrigger="1" onit="0"/>
sp_cli> ex {inRow} {filter}
  <trans>
    <pair>
```

```
      <filterInput ALERI_OPS="u"  a="2" b="a" intData="10"/>
      <filterInput ALERI_OPS="d"  a="2" b="a" intData="20"/>
    </pair>
  </trans>
```

Breakpoint 4 looks for any records received on input with the field "a" equal to 2. Breakpoint 5 looks for any records that produce no output, which means it hasn't affected the state of the filter. If a record results in no output, the filter expression on the output breakpoint is still called once, with `row` set to NULL. You can't compare the NULL values with expressions like `row = null` or `row <> null`; you must use the functions isnull() and isnonnull() instead.

While running the Streaming Processor, it paused on breakpoint 4. The input is an update to a record with a=2. The previous record in the same transaction, with a=1, didn't trigger the breakpoint. Continue running:

```
sp_cli> run
sp_cli> wait_pause
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="4" outSeq="4" stepSeq="12"/>
  <row ALERI_OPS="i"  name="filter" loc="COMPUTE" onbp="4" throttle="512"
    history="100" postSeq="3" inSeq="3" outSeq="2" stepSeq="6"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="4" stream="filter" origin="filterInput"
    expr="row.a = 2" enabledEvery="1" leftToTrigger="1" onit="1"/>
  <row ALERI_OPS="i"  id="5" stream="filter" origin=""
    expr="isnull(row)" enabledEvery="1" leftToTrigger="1" onit="0"/>
sp_cli> ex {inRow} {filter}
  <filterInput ALERI_OPS="d"  a="2" b="a" intData="10"/>
```

Another hit on breakpoint 4, with the deletion of the record with a=2. Continue:

```
sp_cli> run
sp_cli> wait_pause
sp_cli> ex {pause}
  <row ALERI_OPS="i"  name="filterInput" loc="INPUT" onbp="0" throttle="512"
    history="100" postSeq="4" inSeq="4" outSeq="4" stepSeq="12"/>
  <row ALERI_OPS="i"  name="filter" loc="PUT" onbp="5" throttle="512"
    history="100" postSeq="3" inSeq="3" outSeq="2" stepSeq="7"/>
sp_cli> bp list
  <row ALERI_OPS="i"  id="4" stream="filter" origin="filterInput"
    expr="row.a = 2" enabledEvery="1" leftToTrigger="1" onit="0"/>
  <row ALERI_OPS="i"  id="5" stream="filter" origin=""
    expr="isnull(row)" enabledEvery="1" leftToTrigger="1" onit="1"/>
sp_cli> ex {inRow} {filter}
  <filterInput ALERI_OPS="d"  a="2" b="a" intData="10"/>
sp_cli> ex {outRow} {filter}
sp_cli> ex {store} {filter}
  <row ALERI_OPS="u"  a="1" b="a" intData="20"/>
```

This time breakpoint 5 was hit. The deletion of the record with key a=2 has not produced any output. If you look at the store contents, this record wasn't there at the start so it's not necessary to delete it. The store doesn't get updated until the PUT is done, so the store is unchanged when the record was processed. Why wasn't this record in the store before? It didn't pass the filter expression of the `filter` stream — the value of its intData field is too low.

If the Streaming Processor is still running, this is what happens:

```
sp_cli> run
sp_cli> wait_pause
```

There is no activity, because there is no more input data available to the Streaming Processor. The **sp_cli** utility is waiting for the Streaming Processor to run and trigger some breakpoint. This command can't be interrupted in **sp_cli**, but you can kill **sp_cli** itself by entering **Ctrl-C** and then restarting **sp_cli**.

7.5. Notification of the debugger events

As the Streaming Processor changes between running and pausing, single-steps, hits breakpoints and exceptions, and so on, it sends asynchronous notifications to all interested parties. To receive these updates, you can subscribe to the Aleri_RunUpdates stream. This “stream” does not retain any content; its notifications bypass the stream's store, and always have the operations type UPDATE. See the *Authoring Reference Manual* for more details.

The following small example shows the notifications sent out from Aleri_RunUpdates when the above breakpoint example was run on the Streaming Processor:

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
```

The Streaming Processor is set to run, after unconditional breakpoints are created.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="NOBREAK" value="2"
    stream="filterInput"/>
```

The leftToTrigger field of breakpoint 2 decreases but the breakpoint is not triggered.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="0"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="1" stream="filterInput"/>
```

Breakpoint 1 is triggered; the Streaming Processor pauses.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
```

The Streaming Processor is set to run again.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="0"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="2" stream="filterInput"/>
```

Breakpoint 2 is triggered.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="1" stream="filterInput"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="3" stream="filter"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="NOBREAK" value="2"
  stream="filterInput"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="1" stream="filterInput"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="2" stream="filterInput"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="STEP" value="3"/>
```

When the **step trans** command executes, it triggers a number of breakpoints along the way. Eventually it finishes and reports that it has done three steps.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="3" stream="filter"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="STEP" value="3"/>
```

Now **step quiesce** is executed.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
```

The Streaming Processor continues to run.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="0"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="EXCEPTION" value="0"
    stream="filterInput"/>
```

The Streaming Processor pauses when it hits the exception.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
```

The unconditional breakpoints have been deleted, conditional breakpoints have been created, and the Streaming Processor is run again.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="0"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="4" stream="filter"/>
```

The Streaming Processor hits breakpoint 4.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="0"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="4" stream="filter"/>
```

The Streaming Processor continues, but hits breakpoint 4 again.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="0"/>
<Aleri_RunUpdates ALERI_OPS="u"  key="BREAK" value="5" stream="filter"/>
```

The Streaming Processor continues, but this time hits breakpoint 5.

```
<Aleri_RunUpdates ALERI_OPS="u"  key="RUN" value="1"/>
```

The Streaming Processor continues to the end.

These notifications from Aleri_RunUpdates can be used by the automated tools just like the Aleri Studio.

7.6. Examining the data in the Sybase Aleri Streaming Platform

The debugger “examine” commands have been used throughout the examples in this chapter. The **sp_cli**(1) man page provides a complete description of these commands; this section contains general information and some important highlights.

The examination commands work only when the Streaming Processor is paused.

The examination commands return the records in the same format used to send updates to common subscribers. The **sp_cli** command generates them in an XML format. The operation types that occur in “examined” data include not only the standard types seen by the normal subscribers, but also the types that are used exclusively inside the Streaming Processor. These are described in Section 7.1.1, “The Stream Processing Loop”.

There are two ways in which records returned by the examination commands may be grouped:

- Two records may be grouped into an update block. These are printed by **sp_cli** within an XML element named <pair>.
- Multiple records and update blocks may be grouped into a transaction. These transactions are printed by **sp_cli** within an XML element <trans>. If a transaction contains only one record, it's printed as this single record, without the <trans> wrapping.

If a stream employs an input window, as this windows fills, it starts generating SAFEDELETEs for the earlier records. To distinguish these records from the DELETEs sent by the input streams, **sp_cli** prints the pseudo-field `ALERI_RETENTION=1` in each one.

There are three arguments used to choose the data to be examined:

kind
Determines the kind of data to be examined.

stream
Specifies the stream from which the data is taken. To get data from all the streams, leave this field empty.

object
Selects the particular data unit. Use this if there are many units of this kind (for example, variables).

Either the *`stream`* or the *`data`* (or both) can be left empty or omitted if it's not applicable to some kind of data requested in the examine command. The kind and stream must match: Streaming Processor-wide data may not be requested from a stream, and per-stream data may not be requested unless there is a value supplied for the *`stream`* argument.

Some kinds of data are available only from certain streams. For example, the pattern state can be read only from a PatternStream. If the requested kind of data is not available for a certain stream, the error "No such kind of data" is returned, even if this kind of data is supported for other streams.

Some kinds of data may be used both with and without the stream argument. For example, the kind “var” can be used without a stream to examine the global variables and with a stream to examine that stream's variables.

As previously mentioned, the data returned by the "examine" commands may be homogeneous (consisting of records of the same type) or heterogeneous (consisting of records of different types). The examine commands return homogeneous data (most returned data will be homogeneous) within < row> tags.

The groups of data related to the input queues are heterogeneous. Each record gets a tag matching the name of the stream that produced it (records produced from source streams are tagged with the name of the source stream itself). These kinds include: `queue`, `inTrans`, `inRow`, `queueHead`, `queueTail`, `inHist`, `lastInTrans`, `inHistEarliest`, and `inHistLatest`.

There are also heterogeneous groupings of historic data, containing a mix of input and output data. Each of these groups contains one or more pairs of transactions: the first record in each pair is an input transaction, and the second one is the matching output transaction. Each input transaction record is tagged with the name of the stream that produced it; each output transaction record gets the tag `row`. If some input stream records are also tagged with `row`, the only way to tell them from output transaction records is by looking at the order in which they appear.

Some kinds of data deal with history, such as the input transactions processed by the stream and the outputs produced. You can get these data sets separately (as input history and output history) or in mixed mode as described above. When the input and output history are examined separately, the transactions are matched by their index: the first input transaction matches the first output transaction, and so on.

It is helpful to be able to examine the data history when an interesting event happens.

The amount of historical data kept for a stream is determined by the stream's history size setting. This setting can be set globally for all streams, or set for individual streams, using the **sp_cli** command **history**. The default history size limit is 100 transactions. Using large history limits increases the memory usage of the Streaming Processor.

If the Streaming Processor has trace mode off, all history gets discarded, but the limit is kept. The next time trace mode is enabled, the history will start collecting again.

In some cases an empty placeholder is used in a record returned by an examine command. In a previous example, an empty placeholder showed that no output transaction was produced from an input transaction. If there is no ambiguity with transaction boundaries (such as with `outTrans`), **sp_cli** simply returns no data is returned. But in other cases, such as **hist**, a placeholder is included to show that the transaction occurred, but produced no output. A placeholder record includes all fields, including the key fields; the value of each is set to NULL.

Some types of data records use their "natural" field names. Some records, such as "pause", return metadata: the field names in such a record are defined in the Streaming Processor. But other types contain a mix of fields defined by the user and added by the Streaming Processor. In a record of this type, the fields added by the Streaming Processor will have the prefix `Aleri_`. This prefix is reserved and should not be used for user-defined fields.

An example of such mixed fields is `var`, used to examine the variables in computational streams. With a stream, defined in part as:

```
<ComputeStream id="compute" ... >
  <Local>vector(int32) ivec;</Local>
  ...
</ComputeStream>
```

For this stream, the command

```
sp_cli> ex {var} {compute} {ivec}
```

would return data in the following format:

```
<row ALERI_OPS="i"  Aleri_Index="0" Aleri_Value="10"/>
<row ALERI_OPS="i"  Aleri_Index="1" Aleri_Value="13"/>
<row ALERI_OPS="i"  Aleri_Index="2" Aleri_Value="12"/>
```

Here the field name `id` is carried over from the key in the original data row. The field "Aleri_Index", added by the Streaming Processor, contains the index of value per key. The field "Aleri_Value" contains the value itself.

Another interesting example is the data kind aggrGroup, which shows the internal state of the aggregations. Suppose a stream is defined in part as:

```
<!-- Row definition of the aggregation's input -->
  <RowDefinition id="inputRowDef">
      <Column name="a"         datatype="int32" />
      <Column name="b"         datatype="string" />
      <Column name="c"         datatype="double" />
      <Column name="d"         datatype="date" />
      <Column name="intData"   datatype="int32" />
      <Column name="charData"  datatype="string" />
      <Column name="floatData" datatype="double" />
      <Column name="dateData"  datatype="date" />
  </RowDefinition>

  <RowDefinition id="aggregateRowDef">
      <Column name="z"         datatype="int32" />
      <Column name="intData"   datatype="int32" />
      <Column name="charData"  datatype="string" />
      <Column name="floatData" datatype="double" />
      <Column name="dateData"  datatype="date" />
  </RowDefinition>

  <AggregateStream id="aggregate"
          rowdef="aggregateRowDef"
          keys="z"
          ...
          >
      <Group value="input.a" />
      ...
  </AggregateStream>
```

The aggregation state consists of buckets indexed by the aggregation key (in this case the field `z` which originates from the field "a" of the input records). Each bucket contains multiple input rows, which may be sorted or unsorted. When the aggregation status is displayed, there are two types of fields:

- The fields of the key (in this case "z") get `Aleri_Key_` prepended to them, to avoid possible conflicts with the names of the fields in the input rows.
- The field `Aleri_Index` is added to show the position of each row in its bucket.

The command

```
sp_cli> ex {aggrGroup} {aggregate}
```

would produce the output like the following:

```
<row ALERI_OPS="i"  Aleri_Key_z="1" Aleri_Index="0" a="1" b="a" c="1.111000"
  d="2003-12-10 00:00:00" intData="10" charData="aa" floatData="1.111110"
  dateData="2003-12-10 00:00:01"/>
<row ALERI_OPS="i"  Aleri_Key_z="1" Aleri_Index="1" a="1" b="b" c="1.111000"
  d="2003-12-10 00:00:00" intData="10" charData="aa" floatData="1.111110"
  dateData="2003-12-10 00:00:01"/>
<row ALERI_OPS="i"  Aleri_Key_z="2" Aleri_Index="0" a="2" b="b" c="2.111000"
  d="2003-12-11 00:00:00" intData="10" charData="bb" floatData="2.111110"
  dateData="2003-12-10 00:00:02"/>
<row ALERI_OPS="i"  Aleri_Key_z="2" Aleri_Index="1" a="2" b="b" c="2.222000"
  d="2003-12-11 00:00:00" intData="10" charData="bb" floatData="2.111110"
  dateData="2003-12-10 00:00:02"/>
```

In this case, the state contains two buckets with two rows each.

The **sp_cli** command now supports output redirection. For example, piping the examine output to other UNIX commands.

```
sp_cli> ex {aggrGroup} {aggregate} | less
```

7.6.1. Examining with Filters

In certain cases, you may be interested in examining just a few rows from a large volume of data. In the "aggrGroup" example above, you might be interested in one bucket in a stream that could contain thousands. The data returned by the debugging commands can be piped through commands like **grep** to select only the data you want, as in this example:

```
sp_cli> ex {pause} | grep '="filter"'
  <row ALERI_OPS="i"  name="filter" loc="PUT" onbp="5" throttle="512"
    history="100" postSeq="3" inSeq="3" outSeq="2" stepSeq="7"/>
```

Each XML record is printed on a single line; in this display the lines are split. The output of this grep command is a complete record.

But there is a more convenient and efficient way to get a subset of a large data set. The data can be filtered right in the Streaming Processor, before it is even sent out.

The **sp_cli** command **exf** does this filtering, as follows:

```
exf {kind} [ {stream} [ {object} ] ] {expr}
```

The names of the stream and object are optional, just as with the command **ex** and there is an added argument containing the filter expression. The filter expression references a pre-defined variable, with rules similar to the breakpoint filter expressions containing the current row; it compares the row with the expression to decide if the row should be returned. Whenever the filter expression returns true (non-0, non-NULL), the record gets returned to the user and displayed.

The rules for the defined variables are:

- If all the rows in the returned data set are of the same type, they are wrapped in a single variable `row`.

- If the data kind contains rows of the mixed types (the input or history data), multiple variables are defined, with names matching the XML tags printed for these records. At each time only one variable, matching the type of the current record, contains a value. All the others are set to NULL.

These filter expressions filter each row individually, ignoring all transaction and UPDATE_BLOCK boundaries.

Consider an example of selecting a particular aggregate bucket, as defined in the previous example:

```
sp_cli> exf {aggrGroup} {aggregate} {row.Aleri_Key_z = 2}
  <row ALERI_OPS="i"  Aleri_Key_z="2" Aleri_Index="0" a="2" b="b" c="2.111000"
    d="2003-12-11 00:00:00" intData="10" charData="bb" floatData="2.111110"
    dateData="2003-12-10 00:00:02"/>
  <row ALERI_OPS="i"  Aleri_Key_z="2" Aleri_Index="1" a="2" b="b" c="2.222000"
    d="2003-12-11 00:00:00" intData="10" charData="bb" floatData="2.111110"
    dateData="2003-12-10 00:00:02"/>
```

Some data kinds may not support the filtering.

7.6.2. Dumping the Store Data

The debugging tools can also dump the contents of a stream's store into a file, in exactly the same way as the attribute `ofile` in the stream's element causes the stream's contents to be dumped to a file when the Streaming Processor exits and in exactly the same format. If this command is run after the breakpoints example has processed all data:

```
sp_cli> pause
sp_cli> dump {/tmp/debug} {filter}
```

It would create the file `/tmp/debugdump_filter.xml` (the prefix is `/tmp/debug` with `dump_` before the stream name) with the following contents:

```
<filter a="1" b="a" intData="20"/>
```

7.7. Changing the Data in the Sybase Aleri Streaming Platform

The debugging interface can be used to change some data within the Streaming Processor. In **sp_cli**, the **eval** command provides this functionality. Again, this only works when the Streaming Processor is in trace mode and paused.

This command can only change the contents of global and stream local variables (including eventCaches). It cannot change the contents of the stores, queues or working data: that would be too dangerous.

The command changes data by evaluating a SPLASH statement (or block) in the limited context of the stream. The context is limited in the sense that only the variables are visible, not the streams or stream iterators. Any kinds of SPLASH statements may be used, including branching and loops, but writing an infinite loop would hang the Streaming Processor indefinitely. Temporary variables can also be defined inside the SPLASH block.

The eval command changes variables by assigning them as usual.

The operations on eventCaches require special preparation. Normally the key of the eventCache is determined by the current input record. But in this case there is no input record, so the key is not set and any operations on eventCaches would have no effect. For them to work, the key has to be set manually using the operator *keyCache(ec-variable, record)*. It must be set before performing any operations on eventCache. Nothing stops you from changing the key more than once, even in a loop, and thus performing operations on multiple keys.

Only external local (those defined in the XML node <Local>) and global variables may be modified by the eval command. The variables defined inside the SPLASH blocks of a stream exist only when the appropriate methods are run and can't be modified.

The unit of code evaluated is not an expression but a SPLASH statement. It must be either a simple statement terminated by ";" or a block enclosed in braces "{}". Multiple statements must always be enclosed in a block. If braces are used to quote the block argument, the outside quotes don't count as the block delimiters: they are just **sp_cli** quotes.

For example:

```
eval `stream` `a := 1;`

eval {a := 1;}

eval `stream` `{ typeof(input) r := [ a=9; | b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }`

eval {stream} {{ typeof(input) r := [ a=9; | b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }}
```

Bad examples, with incorrect termination or blocking:

```
eval `stream` `a := 1`

eval {a := 1}

eval `stream` `typeof(input) r := [ a=9; | b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r);`

eval {stream} { typeof(input) r := [ a=9; | b= 's1'; c=1.; d=intDate(0);];
  keyCache(s0, r); insertCache(s0, r); }
```

For a more complete example, consider a model with the FlexStream:

```
<RowDefinition id="inputRowDef">
      <Column name="a"         datatype="int32" />
      <Column name="b"         datatype="string" />
      <Column name="c"         datatype="double" />
      <Column name="d"         datatype="date" />
  </RowDefinition>

  <FlexStream id="compute"
      store="store"
      istream="input"
      ofile="output/compute.out"
      rowdef="inputRowDef"
      variables="
          int32 i32 := 999;
          int64 i64;
          double dbl;
          date dat;
          string str;
          money mon;
          timestamp ts;
          [int32 a; | string b; double c; date d; ] rec;
      "
      keys="a" >
      <RowLocalStorage id="s0" type="int32" maxsize="5"/>
      <Method name="inputMethod" stream="input"
          value="{
              [int32 a; | string b; double c; date d; ] record := input;

              i32 := i32 + 1;
              i64 := cast(int64, i32);
              dbl := cast(double, i32);
              dat := intDate(i32);
              str := string(i32);
              mon := cast(money, i32);
              ts := cast(timestamp, i32);
              rec := input;

              record.c := record.c + 9;
              output record;
           }"
      />
  </FlexStream>
```

Start the model with `-DD`, and set and examine some variables:

```
sp_cli> ex {listRls} {compute}
  <row ALERI_OPS="i"  name="s0" type="int32" maxsize="5" maxtime="-1"/>
sp_cli> ex {listVar} {compute}
  <row ALERI_OPS="i" name="i32" type="int32"/>
  <row ALERI_OPS="i" name="i64" type="int64"/>
  <row ALERI_OPS="i" name="dbl" type="double"/>
  <row ALERI_OPS="i" name="dat" type="date"/>
  <row ALERI_OPS="i" name="str" type="string"/>
  <row ALERI_OPS="i" name="mon" type="money"/>
  <row ALERI_OPS="i" name="ts" type="timestamp"/>
  <row ALERI_OPS="i" name="rec" type="[int32 a;|string b;double c;date d;]"/>
```

These are just convenience calls, giving a quick look at what is available. It might be useful for the automated tools, saving them from having to parse the nodes <Local> and <Global> in the XML data model file.

```
sp_cli> ex {var} {compute} {i32}
  <row ALERI_OPS="i"  Aleri_Value="999"/>
sp_cli> eval {compute} {i32 := 0;}
sp_cli> ex {var} {compute} {i32}
  <row ALERI_OPS="i"  Aleri_Value="0"/>
sp_cli> ex {var} {compute} {i64}
  <row ALERI_OPS="i" />
sp_cli> eval {compute} {i64 := 1;}
sp_cli> ex {var} {compute} {i64}
  <row ALERI_OPS="i"  Aleri_Value="1"/>
sp_cli> eval {compute} {i64 := null;}
  <row ALERI_OPS="i" />
sp_cli> ex {var} {compute} {i64}
  <row ALERI_OPS="i"  Aleri_Value="a"/>
```

Here some variables are set, and you can see the changes. Variables were initially set to NULL except for those that were initialized. To examine them, go back to the row with its only field, Aleri_Value, which has the value NULL so it is not displayed. Unlike the result for an empty transaction, the operation type is still INSERT, not NOP. The values of variables can change to something definite or to NULL.

```
sp_cli> eval {compute} {rec.a := 8;}
sp_cli> ex {var} {compute} {rec}
```

This example attempts to set a field in a record variable. But the variable was NULL; any attempts to set fields in a NULL record are silently ignored, so the variable retained its NULL value. Examining record variables containing NULL has a different effect than examining any other variables. Instead of returning a row with all NULL fields, the command returns no rows at all.

```
sp_cli> eval {compute} {rec := [ a=9; | b='s1'; c=1.5; d=intDate(0); ];}
sp_cli> ex {var} {compute} {rec}
  <row ALERI_OPS="i"  a="9" b="s1" c="1.500000" d="1970-01-01 00:00:00"/>
```

Now the record variable is set to contain a record.

```
sp_cli> eval {compute} {rec.a := 8;}
sp_cli> ex {var} {compute} {rec}
  <row ALERI_OPS="i"  a="8" b="s1" c="1.500000" d="1970-01-01 00:00:00"/>
```

And changing a field in an existing record works.

```
sp_cli> eval {compute} {{ Aleri_Key := [ a=9; | b='s1'; c=1.5; d=intDate(0);];
  aggregate(insert, s0, int32, 1); }}
sp_cli> eval {compute} {{ Aleri_Key := [ a=9; | b='s1'; c=1.5; d=intDate(0);];
  aggregate(insert, s0, int32, 2); }}
sp_cli> ex {rls} {compute} {s0}
  <row ALERI_OPS="i"  a="9" Aleri_Index="0" Aleri_Value="2"/>
  <row ALERI_OPS="i"  a="9" Aleri_Index="1" Aleri_Value="1"/>
```

This shows the use of both the RLS and SPLASH blocks.

Even though these examples show evaluation blocks split into multiple lines, each must be entered as one continuous line. In many cases, the multi-line quoting syntax may be more convenient:

```
sp_cli> eval {compute} <<!
'!' to complete> {
'!' to complete>   typeof(input) r := [ a=9; | b= 's1'; c=1.; d=intDate(0);];
'!' to complete>   keyCache(s0, r); insertCache(s0, r);
'!' to complete> }
'!' to complete> !
```

While you are entering a multi-line argument, **sp_cli** displays a different prompt, reminding you to end the argument. You do this by entering a "!" on a line by itself. See the **sp_cli**(1) man page for more details on multi-line quoting.