SYBASE®
An SAP® Company

Integration Guide

Sybase CEP Option R4

DOCUMENT ID: DC01030-01-0400-02
LAST REVISED: April 2011
Copyright © 2011 by Sybase, Inc. All rights reserved.
This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.
To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.
Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.
Sybase trademarks can be viewed at the Sybase trademarks page at *http://www.sybase.com/detail?id=1011207*. Sybase and the marks listed are trademarks of Sybase, Inc. A ® indicates registration in the United States of America.
SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.
Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.
Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.
All other company and product names used herein may be trademarks or registered trademarks of the respective companies with which they are associated.
Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.
Sybase, Inc., One Sybase Drive, Dublin, CA 94568

Contents

Preface 1
Who This Integration Guide Is For 1
How to Use This Guide 1
Integrating Sybase CEP with External Systems 3
Sybase CEP Integration and Enterprise Software Architecture 4
Software Development Aspects of Sybase CEP Integration 5
Other Interfaces to Sybase CEP 5
Other Helpful Documents 5
Data Streams and Messages 7
Definitions 7
Field 7
Message / Row 7
Row Timestamp 7
Schema 8
Tuple Descriptor 8
Data Stream 8
Stream URI 8
Publishing and Subscribing 9
Windows 10
User-Defined Functions 10
Expiring Messages 10
Bundles 11
SDK 12
Adapters 13
Defining Adapter Parameters 13
Streams and Adapters 14
In-process vs. Out-of-process Adapters 15
Sybase CEP Adapters vs. User-written Adapters 16
Creating Your Own Stream Adapter: Overview 16

Key Tasks: Conversion and Communication 16
Conversion 17
Communication 17
Input Adapter Algorithm (Out-of-process) 19
Input Adapter Algorithm (In-process) 19
Output Adapter Algorithm (Out-of-process) 19
Output Adapter Algorithm (In-process) 20
Adapter APIs 20
Starting and Stopping Adapters 20
Engine Control 21
Overview of Engine Control Tasks 21
Definitions 21
Engine Control definitions 21
Query / Statement 21
Project / Query Module 22
Workspace 22
Introduction 23
Commands 23
Creating Streams and CCL Statements from Inside a Program 24
Dynamic Queries and Streams 24
CCL statement background 25
Dynamically Registering Queries and Streams 26
Stopping A Query 33
Troubleshooting the Perl SDK 33
Monitoring Servers and Projects 34
Server status information 34
Status Streams 34
Status APIs 36
User Authentication from Inside SDKs 38
User-Defined Functions and Plugins 39
User-Defined Functions 39
Functionality 39
Requirements 40
Sybase CEP SDKs that Support UDFs 44

Plug-Ins 44
Plug-Ins: Overview 44
Sybase CEP SDKs that Support Plugins 46
Remote Procedure Calls, Database Queries, and Public Windows 47
Remote Procedures 47
Calling a Remote Procedure from a CCL Statement 47
The Components 48
Additional Information about RPCs 52
Generic HTTP and SOAP Plugins 52
Remote Procedure Call Plugin for CSV Files 53
Writing Your Own RPC Plugin 55
Compiling an RPC Plugin 56
Remote Database Queries 56
Overview: Querying a Remote Database or Public Window from a CCL Statement 56
The Components 57
Additional Information 59
Reading and Writing BLOBs on External Database Servers 60
Remote Requests, Synchronization, and Performance 64
Caching 64
Internal Parallelization 64
Public Windows 65
Engine Control: Command-line Tools 67
Engine Control: Command-line Tools: Overview 67
Start the Sybase CEP Server 68
Start the Sybase CEP Server on UNIX-like Operating Systems 68
Start the Sybase CEP Server on Microsoft Windows 69
Access a Running Sybase CEP Server 69
Create a Project and its Associated Schema Files 70

Sybase CEP Project Files 70
Sybase CEP Schema Files 71
Create a Workspace on the Sybase CEP Server 71
Compile a Project or a Schema File 71
Compiling Directly via the CCL Compiler 72
A Note About ADL Files 79
Sending the query module to the server 79
Registering a Project via the c8_client Program 80
Compiling and Running a Project 81
Get Status of an Executing Project 82
Publishing Data to a Server 82
Stop Execution of a CCL Project 85
Clean Up a Workspace's Resources 86
Stop the Sybase CEP Server 86
Stop the Server on UNIX-like Operating Systems 86
Stop the Server on Microsoft Windows 87
Implementing Guaranteed Processing 89
Overview 89
Sybase CEP Engine Application Components 89
Guaranteed Processing Implementation 90
Sybase CEP Engine Settings 91
Settings for Projects, Modules, and Streams 91
Persistence 92
Start with Clean Slate 92
Writing an Adapter for Guaranteed Processing 93
Guaranteed Delivery Mechanisms 93
Guaranteed Delivery with the Sybase CEP C/C++ SDK 94
Guaranteed Delivery with the Sybase CEP .NET3 SDK 96
Variations of Guaranteed Processing 99
Sybase CEP C/C++ SDK 101
C/C++ Overview 101
Compiling for 64-bit Microsoft Windows 102

In-process vs. Out-of-process Activities 102
Data Types and Subroutines for UDFs and In-process Adapters 102
Error Handling Functions 104
Memory Management API 109
C/C++ Data Conversion Functions 111
Miscellaneous C/C++ functions 115
Generic Functions Available in Out-of-process and In-process Tasks 116
APIs Used for Out-of-process Adapters and Control Programs 117
c8client.h 117
APIs Used for In-process and Out-of-process Adapters 118
Schema API 118
Message API 121
Creating an Out-of-process Adapter in C/C++ 141
C/C++ API Interface adapters 141
Creating a Sample Input Adapter 149
Creating an In-process Adapter in C/C++ 157
The Components of an In-process Adapter 158
Algorithm Overview 158
Session State Information and Persistent State Information 159
API Interface 160
Session States vs. Persistent States 174
Signatures of User Functions 176
In-process Input Adapters 178
In-process Output Adapters 181
In-process Adapters 182
Useful Utility Functions 185
Requirements for the C/C++ File 188
Step-by-Step Instructions for Creating an In-process Adapter 189
Compiling an In-process Adapter 190

Practical Tips for Using the In-process Adapter SDK 193
Multi-Stream In-Process Adapters 196
Troubleshooting complier and linker error messages 197
Setting Up Dynamic Queries and Streams with the C/C++ SDK 197
The API 198
Registering a query: Example 202
Creating Streams Dynamically 210
Troubleshooting 210
Control: Compile/Start/Stop/Status 210
Compiling a CCL Project 210
Start/Stop a Project 218
Monitoring Servers and Queries 220
Status API 220
Tracer Message API 233
User-Defined Functions 234
User Defined Functions in C/C++ 234
UDFs: Requirements and Example 235
UDFs: Packing and Unpacking Parameter Values 235
UDFs: Example 236
User-Defined Aggregate Functions in C/C++ 240
UDFs: XML Signatures 242
UDFs: Interface Code 245
Compiling a UDF and Putting It in the Correct Directory 256
UDFs: Summary 261
Querying a Public Window in C/C++ 262
RPC Plugins 264
RPC Plugin API 264
Compiling an RPC Plugin and Putting It in the Correct Directory 267
User Authentication 271

User Credentials API 272
Creating Your Own Authentication Plugin 273
Plugin Configuration 276
Library-Wide Initialization and Shutdown 278
Sybase CEP Java SDK 279
Locating Java files 279
Setting Up Your Environment 279
Using the Java Examples 280
JavaAPI dependencies files in Maven 280
Examining Example 1: Subscribing to a Stream 280
Subscribing to a stream 281
Creating an Engine Client in Java 281
Creating a URI 281
Subscribing to a stream in Java 282
Reading data from a stream in Java 282
Disconnecting from a stream in Java 282
Compiling and Running in Java 282
Other Examples 283
Publishing to a stream in Java 283
Controlling the engine in Java 284
Registering a Query in Java 285
Compiling and starting a project in Java 285
Exploring Value Types in Java 286
Examining Schemas in Java 286
Working with Tuples in Java 286
Retrieving the server status in Java 287
Publishing Asynchronously 287
Subscribing Asynchronously 288
Working with Bundles in Java 288
Guaranteeing Message Delivery in Java 289
Registering a Query with parameters in Java 289
Working with URIs in Java 289
Querying a public window in Java 290
Working with Parallel Queries in Java 290
Sybase CEP .NET3 SDK 291

Locating .NET files 291
Using the .NET3 examples 291
Examining Example 1 292
Subscribing to a Stream 292
Creating an Engine Client in .NET 292
Creating a CCL URI 293
Subscribing to a stream in .NET 293
Reading data from a stream in .NET 293
Disconnecting from a stream in .NET 293
Compiling and Running in .NET3 294
Other Examples 294
Publishing to a Stream in 294
Controlling the Sybase CEP Engine in .NET3 ...295
Registering a query 296
Compiling and starting a project in C/C++ 296
Exploring Value Types in .NET 297
Examining Schemas in .NET 297
Working with Tuples in .NET 297
Retrieving Server Status in .NET3 298
Working with Bundles in .NET 298
Guaranteeing Message Delivery in .NET 299
Registering a Query with parameters in .NET3 . 300
Working with URIs in .NET 300
Querying a Public Window in .NET3 300
Working with Parallel Queries in .NET 301
Sybase CEP Perl SDK 303
Perl prerequisites 303
Perl API Interface 303
Perl API 304
C8::Tuple 304
C8::Publisher 305
C8::Subscriber 305
Perl Input Adapter (Sending Data to a Sybase CEP Stream) 306

Perl Output Adapter (Receiving Data from a Sybase CEP Stream) 307
Installation and Configuration 308
Running the Perl Installation Script 308
Specifying the Path to the Library Files 309
Running the Perl example 310
Troubleshooting the Perl SDK 311
Sybase CEP PythonPython SDK 313
Python API Interface 313
Python API 313
Tuple 314
Publisher 314
Subscriber 315
Python Input Adapter (Sending Data to a Sybase CEP Stream) 315
Python Output Adapter (Receiving Data from a Sybase CEP Stream) 317
Configuring your Python environment 317
Adapter Definition Language 319
Adapter Definition Language: Overview 319
Warnings and Tips 324
Configuring Your System to Find ADL Files 324
Importing New Adapters 326
Server Plugins 327
Message-Driven Plugins 327
Sybase CEP Server Events 327
The Sybase CEP Generic Plugin 334
Implementing Manager HA with the Sybase CEP Generic Plugin 336
Non-Message-Driven Plugins 337
The User Authentication Plugins 337
Datatype Mappings 347
Troubleshooting 351
General Tips 351

Errors When Compiling C-language Adapters, UDFs and RPC Plugins 351
Error Messages When Compiling Java-Language Adapters 352
Errors When Starting the Server 352
Error Messages Displayed During Execution 353
HTTP and SOAP Plugin Configuration 361
Stream URIs 365
What Is a URI and What Is It Used For? 365
Types of URIs 365
Logical vs. Physical Stream URIs 366
Absolute vs. Relative Stream URIs 366
URIs and Distributed Queries 366
URIs and High Availability 367
Finding the URI of a stream 367
Summary 368
Connecting to Streams over a Network 369
Data Stream URI 369
Subscribing to a Data Stream 369
Publishing to a Data Stream 371
Infiniband Support 372
Data Stream Formats 373
Binary Data Stream Format 373
CSV Data Stream Format 373
XML Data Stream Format 375
Sybase CEP Adapters 377
Configuring Adapters 377
Reading and Writing BLOB Data 379
Setting Up the Environment for Java Adapters 379
Setting Up the Environment for the JMS Adapter 380
JMS Adapter prerequisites 380
Configuring and Setting Up Your JMS Server 380
Configuring and Setting Up Sybase CEP 381
Testing the Sybase CEP JMS Adapter 381
Shutdown Sequence 383

Reading, Writing, and Converting Timestamps 384
Adapters Supplied by Sybase CEP 385
Atom Feed Reader Input Adapter 389
Binary: Read From Binary File Adapter 391
Binary: Write To Binary File Adapter. 391
Comma-Separated Values (CSV): Read From CSV File Adapter. 391
Comma-Separated Values: Write To CSV File Adapter. 405
Comma-Separated Values (CSV): Read From CSV Socket Adapter. 409
Comma-Separated Values (CSV): Write To CSV Socket Adapter. 413
Database: Poll From DB Input Adapter 414
Database: Read From DB Input Adapter 420
Database: Write to DB Output Adapter 423
Email: Send Email Out Adapter (SMTP) 464
Java Email Output Adapter 466
Ganglia Input Adapter 469
JDBC Input Adapter 469
JDBC Output Adapter 472
JMS Adapter 476
Random Tuples Generator Adapter 480
RAP - The Trading Edition Adapters 482
Regular Expressions: Read From File Using a Regular Expression Adapter. 485
Regular Expressions: Read From Socket Using a Regular Expression Adapter 489
RSS Feed Reader Adapter. 491
SNMP Get OIDs Adapter 492
SNMP Set Adapter 495
SNMP Send V1 Traps Adapter 496
SNMP Send V2c Notifications 498
XML: Read From XML File 500
XML: Read from XML Socket 504

XML: Write To XML File Adapter. 505
XML: Write to XML Socket 508
XML: Write XML Over HTTP Adapter. 508
Windows Event Log Adapter 509
SNMP Adapter Information 513
Motivation for SNMP 513
Configuring Your Environment for SNMP Adapters 513
Configuring the c8-server.conf File for SNMP 515
Sybase CEP Drivers 517
Status Information 519
Status Information about a CCL Application 519
Status Information about CCL Compiler Settings 522
Status Information for a CCL Query 523
Status Information about CCL Stream Pairs 524
Status Information about a Workspace 524
Status Information about a Container 525
Status Information about a Manager 566
Daylight Saving Time and the Sybase CEP Time Zone Database 567
Time Zones 567
Default Time Zones 568
Daylight Savings Time 568
Transitioning from Standard Time to Daylight Savings Time and Vice-Versa 569
Duplicate Time Zone Abbreviations 569
Sybase CEP Time Zone Database 570
Lengths of Time 570
Date Rules 571
Field Descriptions 572
Index 575

Preface

This document covers a wide variety of topics that are related to interfacing Sybase® CEP with the outside world.

Topics include:

- Data input and output.
- Server control, for example starting and stopping execution of a project (previously called a query module), or getting the status of an executing project.
- Extending the Sybase CEP Server, for example, with User-Defined Functions, which are functions that you write and then can call from CCL statements.
- Embedding the server as a library inside another process.

Who This Integration Guide Is For

This guide is intended for new and experienced users of Sybase CEP, as well as for software architects who plan to use Sybase CEP as part of a larger system and need an overview of how Sybase CEP interfaces with the outside world.

This guide assumes that you have already read the Sybase CEP *Programmer's Guide* and have a general familiarity with the Sybase CEP Engine.

Portions of this guide assume some familiarity with networking and operating system concepts such as sockets and URLs.

How to Use This Guide

This guide is not intended to be read linearly.

The first several chapters of this guide provide conceptual overviews of topics such as adapters, Remote Procedure Calls (RPC), and User-Defined Functions (UDFs) written in C/C++. The next several chapters explain how to write software to do these tasks (for example, write an adapter) using the Software Development Kits (SDKs) that Sybase CEP provides. We recommend that you read the relevant conceptual chapter and then the SDK chapter for the programming language that you plan to use. For example, if you want to write an adapter in Java, then read the chapter that describes the concepts behind adapters and then read the general portions of the Java SDK chapter and the adapter-specific portions of that Java SDK chapter.

If you would like to view sample code, you will find many samples in the "sdk" directory under the "c8/server" directory. On Microsoft Windows systems, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk
```

On UNIX-like operating systems, this directory is typically:

```
/home/<userID>/Sybasec8/server/sdk
```

Under this directory, you will find a subdirectory for each of the languages for which Sybase CEP supplies an SDK (including C/C++, Java 1.5, .NET3, Python, and Perl).

Integrating Sybase CEP with External Systems

This section discusses a variety of topics in the Sybase CEP *Integration Guide* that are related to interfacing Sybase CEP with external systems.

Topics in this manual include:

- Data input and output:
  - Adapters are software that convert data from external formats to the Sybase CEP Server's native format and vice versa. Sybase CEP provides a set of adapters that convert between the server's format and several common data formats. Sybase CEP also provides a set of SDKs to help you build a custom adapter if your data format does not match any of the included adapters or native stream formats.
- Sybase CEP Server Control:
  - Server control allows you to start and stop the execution of a project (query module), check the status of an executing project, and so on.
  - Command-line tools (such as the Sybase CEP CCL compiler) can be used outside of Sybase CEP Studio.
- Extending the Sybase CEP Server:
  - User-Defined Functions, which are functions that you write and then call from CCL statements.
  - Plugins, which allow you to call external programs in response to server status events (for example, such as a server process shutting down).
  - Remote Procedure Calls (RPCs).
- Accessing Sybase CEP streams directly over the network.
- Embedding the server as a library inside another process.

The table below shows the capabilities of each of the Sybase CEP SDKs in terms of converting to different data formats. Some terms used in this table are defined later in the manual; you may wish to re-read this table after you have read about those terms.

Data Format	**C/C++**	**Java 1.5**	**.NET3**	**Perl**	**PythonPython**
In Process Adapters	+				
Out of Process Adapters	+	+	+	+	+
Dynamic Queries	+	+	+		

Data Format	C/C++	Java 1.5	.NET3	Perl	PythonPython
User Defined Functions	+				
CSV Message Format	+		+	+	+
XML Message Format	+	+			
Binary Message Format	+	+			
Operation on Message Bundles	+	+			

Sybase CEP uses the term "Sybase CEP Engine" to refer to the entire product, and uses the term "Sybase CEP Server" to refer to the program that processes queries and generates output. The term "Sybase CEP Engine" thus includes:

- Sybase CEP Server.
- Sybase CEP Studio, which is a GUI tool that helps you create and edit queries, streams, and so on.
- Sybase CEP SDKs, which help you create adapters, user-defined functions, and separate applications that act as "clients" of Sybase CEP Server.
- Sybase CEP command-line utilities, which do tasks such as compiling queries that are written in the CCL query language.

It's common to use Sybase CEP Studio primarily when developing CCL applications, and use the command-line utilities primarily in production. Of course, you may use either Sybase CEP Studio or the command-line utilities in either situation.

Sybase CEP Integration and Enterprise Software Architecture

Use Sybase CEP Engine as a stand-alone product or as part of a larger system. Describes actions you can perform when Sybase CEP Engine is integrated with a larger system.

- Get data into and out of the Sybase CEP Server.
- Exert control over the server (start a query, stop a query, get status of a query, get status of a server) from outside the server.
- Extend the server, for example, by adding new functions that can be called from CCL queries.

- Get data from, or send data to, a database server.
- Embed Sybase CEP Server as a library inside another process.

This document provides information on each of these topics.

The first few chapters of this document focus on concepts. These will tell you how Sybase CEP Engine can connect to the outside world (for example, get data in and out) and how it can be integrated into a larger system. These are the chapters of most interest to architects.

Software Development Aspects of Sybase CEP Integration

An outline of the chapters in the Sybase CEP *Integration Guide* that contain information for developers.

Developers will usually be most interested in the implementation details, such as the following:

- Which languages (for example, C, Java 1.5, .NET3, Perl) Sybase CEP provides SDKs for.
- Which functionality (for example, input/output, control, embedability, User-Defined Functions) is in each SDK.

This document provides information on each of these topics.

The first few chapters of this document focus on concepts. These will tell you what a Sybase CEP Engine is capable of doing. The remaining chapters focus on the "how", for example, how to write programs that will send data to the server or receive data from the server.

If you know that you will focus on a specific task (for example, getting data in and out of the server), then you may skip the detailed "how to" sections for other topics.

Other Interfaces to Sybase CEP

Sybase CEP Server supports other useful interfaces. For example, it provides a status stream that enables you to monitor servers and queries. It also provides an ability to control the Sybase CEP Server with an API based on Simple Object Access Protocol.

Other Helpful Documents

If you are interested in running multiple copies of the server, either to enhance reliability by having backup servers or to enhance performance by doing more parallel processing , see the Sybase CEP *Installation Guide* .

To learn how to read data from (or write data to) database servers and use that data in Sybase CEP Server without writing your own adapter), see the discussion of database subqueries and the Database Statement in the Sybase CEP *CCL Reference Guide* .

Data Streams and Messages

To create adapters (which send data to, or read data from, the server), you need an understanding of data streams and messages.

If you have already read the Sybase CEP *Programmer's Guide* , some of these terms will be familiar to you. This section will list key terms here so that we can expand from simple concepts to more complex concepts.

Definitions

Terminology related to data streams and messages.

Field

A field holds a single piece of data, such as a salary,a person's last name, or a timestamp.

For some data types, such as XML, the content may be more "complex", but to Sybase CEP Server it is still a single piece of data.

Because database tables and Sybase CEP windows can be thought of as arranging data in rows and columns, we use the terms "column" and "field" interchangeably.

Note that a field may contain a NULL value to indicate that the value is unknown.

Message / Row

A message(also called a "row", a "tuple", or an "event") is very similar to a row of data in a database table. A row is a group of fields that all have data about the same thing.

For example, a row may describe a single stock trade, and may contain the price at which the stock was sold, the number of shares sold, and the stock name or stock symbol. If there are multiple stock trades, each will be described in a separate row.

Most messages match the description given above. However, there are also "control" messages (as opposed to "data" messages). Control messages are described later.

Messages "travel" in a stream. For more information, see *Data Stream* on page 8.

Row Timestamp

In a Sybase CEP Engine, each row contains a "row timestamp" that is used to help ensure that rows are processed in order.

If the row contains information about a particular event (such as a stock trade), the row timestamp is typically the date and time that the event occurred (such as the time that a stock

trade was completed). Row timestamps are stored using the Sybase CEP TIMESTAMP data type, which means that the time information is stored with a precision of one microsecond.

Row timestamps are not necessarily unique. Two rows (in the same stream or in different streams) may have the same row timestamp.

Schema

Every row in a stream has the same structure, or "schema". The schema comprises the column names, the column data types, and the order in which the columns appear.

For example, in a StockTrades stream, the first column might be a STRING that contains a stock symbol, the second column might be a FLOAT that contains the price per share, and the third column might be an INTEGER that contains the number of shares bought or sold.

Multiple streams may use the same schema. One stream may not use multiple schemas.

The schema defined by the user does not explicitly include the row timestamp, but all rows will have a row timestamp as a "hidden" field.

Tuple Descriptor

A "Tuple Descriptor" stores information about the schema of a stream and information about the schema of the rows in the stream. Adapters written using certain Sybase CEP SDKs use a Tuple Descriptor to help the adapter work properly with the stream.

A Tuple Descriptor is created by compiling a file that contains the schema definition.

Tuple Descriptors are documented in more detail later in the manual, when we document how to publish and subscribe to streams.

Data Stream

A data stream carries rows from one place to another, for example, from one query to a subsequent query, or from an adapter to the server.

A stream is associated with a particular schema, and every row in the stream must match that schema.

Stream URI

Each stream has a unique URI, which serves as anaddressthat enables software to connect to that stream. For example, when an adapter connects to a stream, the adapter needs to know the stream's URI.

Some sample URIs are below:

- `ccl://localhost:6789/Stream/Default/Stocks/InStream1`
- `ccl://server1.admin.sybase.com:6790/Stream/Default/RFIDTracers/InStream1`

A stream URI may be either a "physical" URI or a "logical" URI:

- A "physical" URI begins with "http://" and refers to the name or IP address of a particular machine.
- A "logical" URI begins with "ccl://" and must be translated into a physical (HTTP) URI before a connection can be made.

Most of the time, users will use CCL URIs.

To see the URI of a stream, go into Sybase CEP Studio, click on the stream, and then select the "Properties" tab for the stream. The tab will specify the "Stream URI" (the logical URI) and the "HTTP URI" (the physical URI).

See *Stream URIs* on page 366 for a more detailed discussion of HTTP and CCL URIs.

Publishing and Subscribing

Messages are written to a stream and then read from that stream.

When a message is written to a stream, we call that "publishing". The publisher may be a CCL query, which writes the output of the query, or an adapter, which gets data from an external data source and then writes that data to the stream.

When a query or adapter connects to a stream to read the messages sent on that stream, we call that "subscribing".

Each stream may have zero or more subscribers. Each subscriber to a stream receives a copy of all messages sent to that stream from the time that the subscriber subscribes to the stream until the time that the subscriber unsubscribes. Unless the Guaranteed Delivery feature is used, messages published before the subscriber subscribed are not sent to the subscriber.

Each stream may have zero or more publishers. If there are multiple publishers, and if the publishers specify the row timestamps in the rows, then the rows need to be synchronized, in other words, arrive at the stream's destination in the proper order. There are at least two ways to do this. If the publishers synchronize with each other, they can simply insert the rows in the proper order. If the publishers do not synchronize with each other, set the appropriate stream properties ("Maximum Delay", "Messages can come out of order", and "Maximum out of order delay") to tell the Sybase CEP Server to sort the incoming messages and process them in order.

Note that synchronization applies both within a stream (if there are multiple publishers to that stream) and across multiple streams.

For more information about stream synchronization and out-of-order messages, see the Sybase CEP *Programmer's Guide* .

The most common way to publish or subscribe to a stream is to create an adapter using one of the Sybase CEP SDKs.

It is possible to use the URI to publish directly or subscribe directly to a stream over the network without using an adapter. See *Connecting to Streams over a Network* on page 369.

Note that you can only read and write to a stream while the project (query module) that contains it is running.

Windows

A window is a stored group of rows that has a retention policy, such as "KEEP 10 MINUTES" or "KEEP 50 ROWS".

A query may operate on one or more windows, as well as on a stream. This enables you to relate a new row to rows that arrived earlier. It also allows you to perform aggregate functions (for example, SUM(), AVG(), and so on) on all the data accumulated in a window.

Below is an example of a query on a window.

```
-- Create the window.
CREATE WINDOW StockTradeHistory
SCHEMA ...
KEEP 8 HOURS;
...
-- Query: Process data from "history" window and generate output.
INSERT INTO OutputStream...
SELECT StockSymbol, AVG(Price)
FROM StockTradeHistory;
```

Unless the window is defined with a KEEP ALL clause which allows it to accumulates messages indefinitely, rows from the window will eventually expire and be deleted from the window.

User-Defined Functions

Write your own functions and call them from CCL; these functions are called User-Defined Functions (UDFs). As with built-in functions, you can pass in values from the current row, and you get back a return value from the function.

You may write UDFs in either C/C++ or by using CCL's CREATE FUNCTION statement. Only C/C++ UDFs are discussed in the Sybase CEP *Integration Guide* . For information about the CCL CREATE FUNCTION statement, see the Sybase CEP *CCL Reference Guide* .

C/C++ UDFs are described in more detail later in this manual; see *Sybase CEP C/C++ SDK* on page 101.

Expiring Messages

When rows are deleted from a window, some queries may need to be re-evaluated.

If you have a query that calculates the sum of the prices in a window, then when a row expires, the price in that row should be subtracted from the sum of the prices in the window. Thus if you have a query on a window, each row that arrives in that window may generate two output messages: one for the arriving row and one for the expiring row.

Let's look at an example. Here, two windows are created and a set of queries is created when the output of one window is fed to another window:

```
CREATE WINDOW StockTradeHistory
SCHEMA ...
KEEP 8 HOURS;
-- Query #1: Insert source data into "history" window.
INSERT INTO StockTradeHistory
SELECT *
FROM NYSEStockTrades;
-- Query #2: Process data from "history" window and
-- generate output.
INSERT INTO OutputStream...
SELECT StockSymbol, AVG(Price)
FROM StockTradeHistory
KEEP LAST PER StockSymbol GROUP BY StockSymbol;
```

As you can see, the output of the last query depends upon the contents of the StockTradesHistory window. When a new row arrives in the NYSEStockTrades stream and is put into the StockTradeHistory window, we should calculate a new average price, and when a stock trade's information expires from the StockTradeHistory window, the window will calculate a new average price based on the remaining rows in the window.

If this window directly feeds another window, then when a row from this window expires, the window sends a "delete" message to the second (downstream) window, telling that window to delete its copy of the row.

When one window sends another window a "delete this row" notice, we refer to the notice as a "negative tuple" (because it cancels out the original "positive tuple").

When a User-Defined Function is called from a CCL statement, the function may be passed either a positive tuple or a negative tuple. If you write your own output adapter to subscribe to a stream, your adapter may receive negative tuples and will need to handle them properly. In many cases, all you need to do is determine that the tuple is a negative tuple and then discard it, but in some cases your adapter, like a query, may want to pass on or act upon that negative tuple. To learn how your adapter can recognize that a tuple is negative, see *User-Defined Aggregate Functions* on page 41 in the C/C++ SDK chapter.

Bundles

In some cases, a particular event can cause multiple output rows, and those rows can be grouped into a bundle. A bundle is a set of two or more data messages that are conceptually part of the same operation and are thus grouped together.

Your output adapter may see and process bundles differently from individual messages.

Bundles apply only to adapters, and not to UDFs.

There are two situations in which bundles may occur:

- Bundles can occur when you have a query that joins a stream to a window. In some cases, a row that arrives in the stream may match multiple rows in the window, and for each of those

matches the join query will produce an output row. In that case, all the output rows for a given input row will not only have the same row timestamp, but also will be considered part of a "bundle", since they were all generated by the same event (the same arriving message/row).

- Bundles can also occur when you use the XML "shred" command. A single XML value may contain multiple elements, and some operations on that XML value may generate multiple rows (one per element). In this situation, all the rows generated from that one XML input value will be bundled together.

Bundling is not the same as aggregating. An aggregate function, such as AVG() or SUM(), produces a single output from multiple input rows. For example, if you have 100 stock transactions and thencalculate the average price, you will onlyonly receive oneoutput (the average price). If you perform an operation that creates a bundle, that bundle can have multiple rows instead of just one, even though the entire bundle was created based on a single event or arriving row. Each of the rows in a bundle is a discrete row, and any query downstream will see (and process) each of those rows.

Although all messages in a bundle have the same row timestamp, not all messages with the same row timestamp are necessarily in the same bundle.

A bundle may contain any type of message (for example, negative tuples as well as positive tuples) EXCEPT other bundles. In other words, there is no such thing as a nested bundle.

In some situations, your output adapter may need to recognize that messages have arrived as a bundle and may need to extract individual messages and possibly process the bundled messages in a special way.

The details of extracting messages from a bundle are specific to each SDK (for example, the Sybase CEP Java SDK vs. the Sybase CEP C/C++ SDK) and are discussed in more detail later.

SDK

A Software Development Kit (SDK) is a set of software that helps you write other software, such as input and output adapters and User-Defined Functions.

Sybase CEP supplies SDKs for various programming languages, including C/C++, Java 1.5, .NET3. This guide contains a chapter about each of these SDKs.

Adapters

This section provides an overview of adapters.

Defining Adapter Parameters

You can define adapter parameters in CCL/CCX or in runtime configuration files, as well as via the Sybase CEP Studio ADL files.

Only one adapter configuration should be defined per configuration file. The macro $RuntimeFolder can be used to path to the runtime configuration files.

The value of this macro is a directory called "runtime" under the directory defined by the "StoragePath" preference under the C8/Server/Common section in the server configuration file. See *Stream Binding Runtime Configuration* for more details.

You can turn on runtime configuration from within Sybase CEP Studio, at which point you must define the name and location of the configuration file.

All adapters have an optional property that can be set. The "Runtime property file" allows for the setting of a file name. The file may be pathed to with either an absolute path or the $RuntimeFolder macro. Values in the property file will override any default values set in the Adapter ADL file or explicitly set in the property page within Studio. The property file format is a standard property file format. The property names are identical to those that are written to the CCL in Sybase CEP Studio for a given adapter. Whereas in Studio, the property value shows up as being in quote in the CCL, in theproperty file, there should be no quotes.

For example, in CCL this would look like:

```
ATTACH INPUT ADAPTER ...TO STREAM ...PROPERTIES
PROP1 = "prop1 value"
PROP2 = "prop2 value"
PROP3 = "prop3 value"
```

Whereas in the property file, this would look like:

```
PROP1 = prop1 value
PROP2 = prop2 value
PROP3 = prop3 value
```

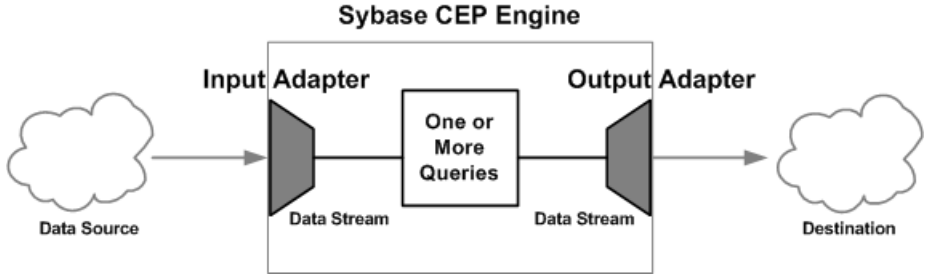
Streams and Adapters

An input adapter reads data from an external data source, and puts the data into a Sybase CEP stream. The input may be read from a file, a socket, a database server, or an RSS feed.

An input adapter reads data from an external data source, and puts the data into a Sybase CEP stream. The input may be read from a file, a socket, a database server, an RSS feed, and so on.

An output adapter reads data from the Sybase CEP output stream, converts the data to an appropriate external data format, and writes the data to the data destination. The output may be written to a file, a socket, a database server, an email program, and so on.

The basic flow of data in a Sybase CEP Engine is thus:



Sybase CEP provides adapters that will read data from an external data source and write it to a Sybase CEP stream (or read from a Sybase CEP stream and write to an external data destination). For example, the ReadFromCSVFile adapter (supplied by Sybase CEP) reads data from a file of Comma-Separated Values (CSV) and publishes the data to a stream. The Sybase CEP Server does not directly read data from the input file; the server reads only from data streams. The adapters supplied by Sybase CEP are described in *Sybase CEP Adapters* on page 377.

Some input and output adapters supplied by Sybase CEP can be used from inside Sybase CEP Studio. To use these, you typically do the following:

1. Right-click on the stream to which you wish to attach an adapter.
2. Choose "Attach Input Adapter" or "Attach Output Adapter".
3. Select the appropriate adapter type (such as "Read from CSV file").
4. Fill in the appropriate parameters for that adapter. For example, if you attach the adapter that reads from a Comma-Separated Values (CSV) file, you will specify the name of the file to read from.

   If none of the adapters supplied by Sybase CEP will fit your needs, you may write your own adapter. Your adapter, like a Sybase-supplied adapter, will read (or write) data in a

format appropriate for your application, and publish the data to (or read the data from) the Sybase CEP Server.

Sybase CEP provides SDKs (Software Development Kits) that allow you to write input and output adapters in the following languages:

- C/C++.
- Java 1.5.
- .NET3 (supported on Microsoft Windows only).
- Python.
- Perl.

In-process vs. Out-of-process Adapters

An adapter can run as either a separate process or as part of the Sybase CEP Server. An adapter that runs as a separate process is called an Out-of-process adapter. An adapter that runs as part of the server is called an In-process adapter.

Note: Out-of-process adapters were previously called "Unmanaged adapters", and In-process adapters were previously called "Managed adapters"; you may still see those terms used occasionally.

In-process adapters generally run faster because the server can get data from (or to) the adapter with less overhead. In-process adapters are started by the Sybase CEP Server when the server starts the corresponding project (query module). The adapter is recognized by Sybase CEP Studio, and from inside Sybase CEP Studio you may attach the adapter to a stream by selecting the adapter from a drop-down list of adapters. A disadvantage of in-process adapters is that if the adapter crashes, it will probably bring down Sybase CEP Server.

Out-of-process adapters deliver rows more slowly than in-process adapters do, are not recognized by Sybase CEP Studio, and must be started manually (rather than by the server), but are safer because an error that causes the adapter to crash will not take down Sybase CEP Server with it. Out-of-process adapters can run on a different computer from the computer that the server runs on.

Currently, in-process adapters can be written only in C/C++. You can write out-of-process adapters in other languages, as well as C/C++.

	In-Process Adapter	Out-of-Process Adapter
Speed of row delivery	Generally faster because the server can get data to and from the adapter with less overhead.	Generally slower.
Started by	Server	User
Recognized by Sybase CEP Studio	Yes	No

Can a crash in the adapter cause Sybase CEP Server to crash	Yes	No
Can run on a different computer than Sybase CEP Server	No	Yes
Languages	C/C++	C/C++ Java 1.5 .NET3 Python Perl

Sybase CEP Adapters vs. User-written Adapters

Sybase CEP provides several SDKs that enable you to write an adapter.

If these do not meet your needs, you may write your own adapters. Sybase CEP provides several SDKs to enable you to write an adapter. These SDKs include:

- C/C++.
- Java 1.5.
- .NET3.
- Perl.
- Python.

All of these SDKs allow you to write out-of-process adapters. Currently, in-process adapters can only be written in C/C++.

Creating Your Own Stream Adapter: Overview

This section provides an overview of how to write your own stream adapter. You may need to write your own adapter if you have a data source that is not supported by the adapters provided by Sybase CEP.

Key Tasks: Conversion and Communication

Adapters perform two key tasks: conversion and communication. Adapters convert data from sensors or embedded systems, convert data read from or written to database servers, read log files, and so on.

An input adapter reads data from a source that is external to the Sybase CEP Engine (for example, from a data feed, from a message bus, from a file, from a database server, or from a sensor), converts that data into a representation appropriate for a particular SDK being used, and publishes the data to a stream by using appropriate SDK function calls. The stream, in turn, sends the data to the engine.

Similarly, for output, the engine sends data to a stream; the stream sends the data to an output adapter; and the output adapter converts the data to the desired external format and writes it to a data destination, such as a file.

The basic flow of data in a Sybase CEP Engine is thus:

input adapter -> stream -> engine -> stream -> output adapter

Adapters may do such things as: converting data from sensors or embedded systems, converting data read from or written to database servers, reading log files, and so on.

Conversion

Your adapter must convert between the data format(s) appropriate for your application and the representation of rows as defined by the SDK you are using.

You should be familiar with your application's data formats, and you should be able to extract the data into rows and fields.

If your adapter is an input adapter, then once you receive a row of data, you call SDK functions to construct a new row, set its timestamp, set its fields, and publish it to the stream.

If your adapter is an output adapter, then once you receive a row of data, you call SDK functions to read fields from that row and send the information to your data destination.

Communication

Adapters communicate by sending data to or reading data from a stream.

Out-of-process Adapter

An out-of-process adapter must perform several actions to send data to or receive data from the server.

1. Acquire an "address" (URI) that will uniquely identify a specific stream and tell the communication layer (provided by Sybase CEP) how to find that stream. The address may be hard-coded in your program or passed as a parameter.
2. Open a connection to the stream. The SDKs provide methods to do this.
3. Write (or read) the desired data. Typically the write (or read) operation is in a loop; the adapter will keep multiple rows of data. The SDKs provide methods to send and receive rows.
4. Close the connection.

Out of process adapters communicate with the Sybase CEP Server through the network layer.

Note: If the project (query module) is not running, then the stream won't exist, and the out-of-process adapter won't be able to connect to it.

Subscribing with Filters (Predicates)

If you are subscribing to a stream or master window that has been defined with the FILTERCOLUMNS property, you can append a query to the end of the URI to specify values

for those columns in the subscription. With this kind of subscription, you only receive rows that have values in the filter columns that match the values you specify in the query. The format is stream_uri?X-C8-Filter=value1, value2, ..., valueN. The values you specify here are for the columns listed in the FILTERCOLUMNS property of the stream or window definition, and must be in the correct order.

For more information about the FILTERCOLUMNS property, see "Create Stream Statement" and "Create Window Statement" in the Sybase CEP *CCL Reference Guide* . See the specific SDK references elsewhere in this guide for information about the functions and methods available. Some of the SDKs have convenience functions to append the query string to a URI, while others simply accept a URI with the appended query as the parameter to the function that establishes the subscription.

If you are subscribing to a master window and want to mirror the contents of that window, you'll need to watch for negative messages (tuples) and remove the existing positive message with an ID that matches that of the incoming negative message.

Subscribing with Filter Expressions (Arbitrary Predicates)

You can subscribe to any stream or master window and filter the results, even if the stream or window has not been defined with the FILTERCOLUMNS property. With this kind of subscription, you only receive rows for which the expression you specify in the query evaluates to True. The format is stream_uri?X-C8-FilterExpr=expression, where expression is a standard CCL Boolean scalar expression, such as the following:

```
ccl://localhost:6789/Stream/..../Foo?X-C8-FilterExpr=Symbol%3dIBM
+AND+Price>10
```

Note that the expression is URL-encoded ("%3d" is a URL-encoded equal sign, for example), which may be handled for you, depending on the SDK call you use.

Also note that you cannot include any of the following in the expression:

- Aggregator functions, such as SUM() or COUNT().
- Stateful operators like PREV().
- FIRST / LAST / INDEX operators.
- CCL variable references.
- GETTIMESTAMP().
- GET__COLUMNBYNAME().
- XMLPATTERNMATCH().
- Functions used within XMLTABLE().

However, you can include user-defined scalar functions and the zero-argument variant of GETTIMESTAMP().

You can also combine filters and filter expressions when subscribing to a stream or master window defined with the FILTERCOLUMNS property, as in the following:

```
ccl://localhost:6789/Stream/..../Foo?X-C8-Filter=NYSE,IBM&X-C8-
FilterExpr=Price>10
```

See the specific SDK references elsewhere in this guide for information about the functions and methods available. Some of the SDKs have convenience functions to append the query string to a URI, while others simply accept a URI with the appended query as the parameter to the function that establishes the subscription.

In-process Adapter

In-process adapters communicate with the server via function calls.

Input Adapter Algorithm (Out-of-process)

A description of how the algorithm of a typical out-of-process input adapter operates.

1. Get the URI of the stream to write to.
2. Connect to that stream.
3. Connect to the input data source (this is specific to the application).
4. Read a row of data from the data source.
5. Instantiate a "row" (tuple/message) and fill its fields with the values received.
6. Send the row.
7. Disconnect.

Steps 4-6 are typically done in a loop that is executed once for each incoming row of data.

Note that if the project (query module) is not running, then the stream won't exist, and the out-of-process adapter won't be able to connect to it.

Input Adapter Algorithm (In-process)

Currently, in-process adapters can only be written in C/C++.

An overview of writing an in-process adapter in C is in *Algorithm Overview* on page 158.

Output Adapter Algorithm (Out-of-process)

A description of how the algorithm of a typical out-of-process output adapter operates.

1. Get the URI of the stream to read from.
2. Connect to that stream.
3. Connect to the output data destination (this is specific to the application).
4. Wait for messages (rows) from the input stream.
5. When you receive a message, read the data and write whatever is appropriate to the output destination.
6. Disconnect.

Output Adapter Algorithm (In-process)

Currently, in-process adapters can only be written in C/C++.

An overview of writing an in-process adapter in C is in *Algorithm Overview* on page 158.

Adapter APIs

Sybase CEP provides Application Programmer Interfaces (API) that enable you to implement in-process and out-of-process adapters.

1. In-process adapters in C/C++.
2. Out-of-process adapters in C/C++.
3. Out-of-process adapters in Java 1.5.
4. Out-of-process adapters in .NET3.
5. Out-of-process adapters in Perl.
6. Out-of-process adapters in Python.

Additional APIs for other programming languages may be added in the future.

Subsequent chapters describe example adapters and instructions for compiling and using them.

Starting and Stopping Adapters

In-process adapters are started automatically by the server. Out-of-process adapters must be started by the user.

The order in which adapters are started is important. Since adapters connect to streams, the adapters must be started after the streams already exist (in other words, after the project (query module) has started), or the adapters must be able to handle the error that they get when the stream doesn't exist and must then wait and re-try to connect to the stream. In addition, if the input adapter starts inserting data before the output adapter is ready, some of the data may "disappear" (never show up in the output). Start projects and adapters in the following order:

1. Project (query module).
2. Output adapter.
3. Input adapter.

   If you have written your output adapter to re-try to connect to the stream, then you can start the project and the output adapter in either order.

   If you are using Guaranteed Delivery, you can start the queries and adapters in any order. However, to minimize the chance of overloading any buffers, it is best to use the order shown above.

Engine Control

This chapter explains the basics of controlling the Sybase CEP Engine and introduces the use of CCL.

Overview of Engine Control Tasks

This section outlines various Engine control tasks, including:

- Creating a workspace in which you can load a query module.
- Deleting a workspace.
- Executing a CCL program.
- Dynamically adding queries and streams.
- Stopping the execution of a CCL program.
- Monitoring servers and queries.

These can be done with Sybase CEP Studio, with command-line tools, and with an API in certain Sybase CEP SDKs. Before explaining how to do these tasks, we will define some terms and illustrate some concepts.

Definitions

Provides definitions for important aspects of Engine Control.

Engine Control definitions

This section contains definitions of key Engine Control terms.

- *Query / Statement* on page 21
- *Project / Query Module* on page 22
- *Workspace* on page 22

Query / Statement

Sybase CEP queries are written in the Continuous Computation Language (CCL), which is based on the SQL language used in many database servers.

This manual uses the terms "query" and "statement" interchangeably.

Project / Query Module

CCL statements are stored in files with the extension ".ccl". Each .ccl file is called a "query module".

When a query module is compiled and executed, all of the statements in the query module are compiled together and stored in the same file (which has the extension ".ccx"). The server executes the .ccx file as a whole, so all the statements in the query module are executed together.

Query modules may be re-used simply by loading them multiple times. For example, if you have query modules A, B, and C, you may load a copy of C into both A and B. (If you load more than one copy of module C into module A, you will need to use a different name for module C each time that you load it into A. If you are using Studio, Studio will automatically append a unique suffix (for example, "_1", "_2", and so on) each time that you load an additional copy of A into B.)

Warning! Because there is still only one underlying module, even if you load multiple copies of it, changes to one instance of the module will also affect the other instances.

A query module that is not nested inside any other query module is called a *project*.

A module nested inside another module is called a *sub-module*.

If a project contains other modules, then when the project is compiled, all of the modules within it are compiled at the same time and their compiled code is stored in the same .ccx file as the project's code.

A project is sometimes called a program or application; the terms are synonymous.

Workspace

Before you can run a program, you must load it into a workspace. Workspaces enable you to create groups in which all query modules are related in some way.

The grouping is up to you. For example, groups can be created based on projects, or on individual developers, or on criteria you choose.

Warning! If more than one person loads the same query module into Studio, makes changes and then saves it, the last person to save will overwrite any changes that other people made. Using separate workspaces does NOT prevent this problem. Workspaces provide separation within the server, but not within the file system. Use a source code control system to avoid conflicting changes to a query module.

Introduction

New users typically use Sybase CEP Studio to create, compile, load, start running, or stop running a project.

You can also perform these tasks using command-line utilities, or from inside a program by calling API functions provided by some of the Sybase SDKs.

The following environments provide some or all of the interface functionality needed to deploy applications on Sybase without using Sybase CEP Studio:

- Command-line tools (c8_server, c8_compiler and c8_client).
- Java 1.5 SDK.
- Microsoft .NET3 SDK .
- C/C++ SDK.
- Perl SDK.
- Python SDK.

Commands

You can use other development environments, or you can work directly from the command line to build applications.

Most developers who build Sybase CEP applications will use the to create and execute query modules on Sybase CEP Server. provides an easy-to-use graphical user interface for editing, debugging and testing queries. Sybase CEP Studio automatically interfaces most of the various components of the for the developer.

However, some developers may need to create and deploy their Sybase CEP applications from either another development environment (such as a scripting or programming language) or from the command line. Although these approaches require more detailed knowledge of the interaction between the components, the flexibility of Sybase CEP Engine may offset this disadvantage.

In order to deploy a application, you must be able to do the following:

- Start Sybase CEP Server.
- Access a running .
- Create a workspace on the Sybase CEP Server.
- Create a query module and its associated schema files.
- Compile the query module using the Compiler.
- Execute the query module on the Sybase CEP Server.
- Stop the query module on the .

- Stop the Sybase CEP Server and clean up its resources.

These tasks are executed using programs described later in this manual. Some SDKs include functions or methods to do some of these operations, such as compiling and executing a query module.

Creating Streams and CCL Statements from Inside a Program

This section explains how to create streams and CCL statements from inside a program.

Dynamic Queries and Streams

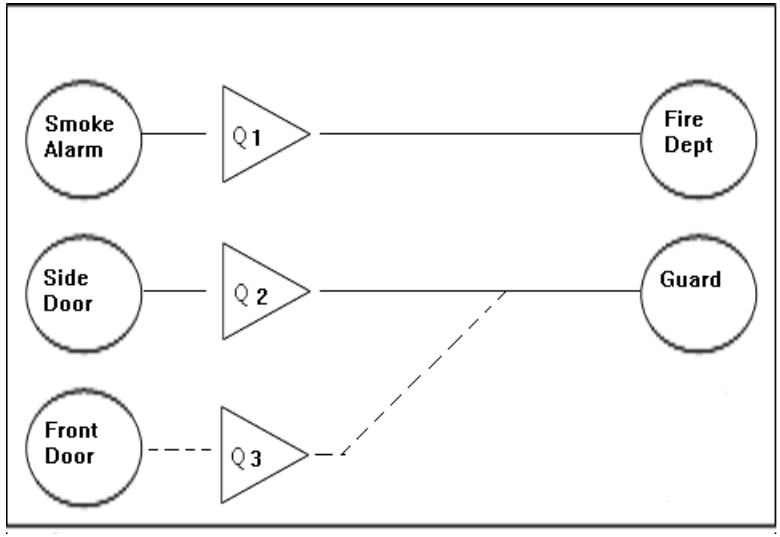
An overview of when to create and run dynamic queries and streams.

The ability to dynamically run or stop CCL statements and streams is useful if you have a system that is not static. For example, consider an application that monitors building security. The input events may come from door sensors, key-card readers, and smoke detectors, while the output events might include alerts to the security personnel or the fire department. Different queries are required on work days than on non-work days, and some queries need to run continuously while others should be activated or deactivated (automatically or via user intervention) based on the time of day and the day of the week.

The building security application might contain the following:

- Smoke detector alerts: Monitors the smoke detectors and send an alarm to the fire department if smoke is detected. The query(s) and streams related to smoke detectors are not dynamic; they are never "turned off".
- Side-door alerts: During all hours, an alert is sent to the on-site security guards if any "side doors" (any doors other than the main entrances) are opened without the use of a valid electronic badge.
- Front-door alerts: During non-working hours, an alert is sent to the on-site security guards if a front door is opened without the use of a valid electronic badge. This alert is not sent during working hours.

We can model this system as shown below. There are 3 input sources, 3 queries, and 2 destinations for alerts:



Sybase CEP Studio can be used to create a project that contains the Smoke Alarm input stream, the Fire Department output stream, and query Q1. The project can then be started and left to run indefinitely.

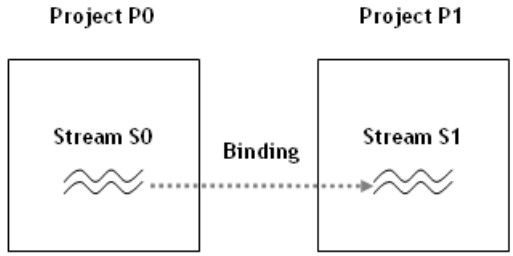
A project can also be created to have the Side Door input stream, the Guard output stream, and query Q2 running continuously.

To have the Front Door input stream and the query Q3 run only outside of normal working hours, a separate program can be used to register (create and run) them at the appropriate times. These streams would stop running during periods when they are not needed.

CCL statement background

Using Sybase CEP Studio to create a project that contains CCL statements and streams.

A CCL statement in one project (for example, project "P1") may operate on data produced in another project (for example, "P0") and send data to yet another project (for example, "P2"). When you use Sybase CEP Studio to set up data transfers between projects, you bind a stream in one project to a stream in another project. If the streams are in the same project, binding merely means providing another alias for one stream. For example, if project P1 wants to read from stream S0 in project P0, then you'll create a stream S1 in project P1 and bind stream S1 to stream S0.



After you've bound the streams, any CCL statement in project P1 that wants to read stream S0's data can simply read from stream S1 to get the data.

The same types of operations (creating CCL statements, creating streams, binding streams, and then running those statements and streams) may be done with eitherSybase CEP Studio or with a program that you have written. This is called "registering a query" , and is conceptually similar to creating a project in Sybase CEP Studio.

In the rest of this section, we will give a conceptual overview of how to register a query. The specific details, including the functions or methods that you call to register a query in a particular SDK, will be explained in the chapters that document the APIs. Currently, query registration is supported in the following SDKs:

- Sybase CEP C/C++ SDK.
- Sybase CEP Java 1.5 SDK.
- Sybase CEP .NET3 SDK.

The function or method that registers a query is generally referred to as the "registerQuery() method", although the exact name may vary depending upon which SDK you are using.

Dynamically Registering Queries and Streams

A description of how to dynamically register queries and streams.

Registering a query includes:

- Creating the streams specified in the registered query.
- Creating the statements specified in the registered query.
- Compiling the CCL statements.
- Loading the query into the server (the query name and the workspace that the query goes into are specified as part of the `registerQuery()` call).
- Connecting ("binding") the stream names in the CCL statements to the existing streams (in other words, to the URIs of those streams), or creating new streams. Users can also specify a file name instead of a stream, in which case the content of the file is used as the runtime URI. See *Stream Binding Runtime Configuration* on page 29 for more details. Users can also create and register queries through the SDK using the binding option on the

c8_compiler. This is executable when working with project and CCL files. See *Compile a Project or a Schema File* on page 71 for more details.

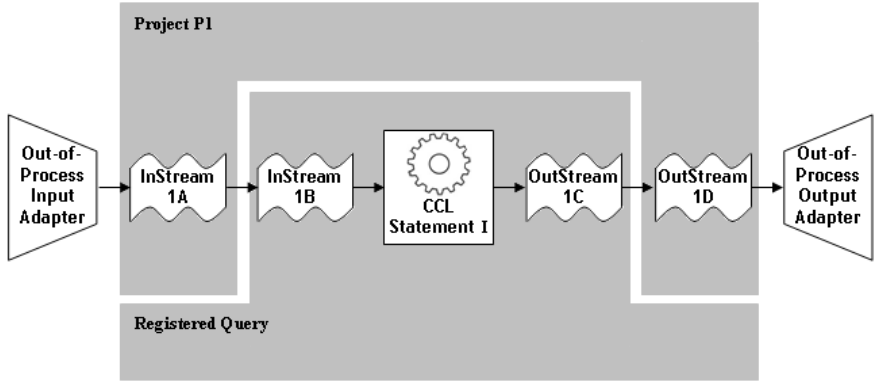
- Starting execution of the query.

The query name must be unique within the specified workspace. The registered query is uniquely identified by the workspace name and query name, both of which are passed as part of the `registerQuery()`call.

A registered query may be stopped later.

There is no separate "start" method/function call. The query is started at the time that it is registered. After you stop it, there is no start/restart call; the only way to restart it is to re-register it. If you stop the query before you re-register it, you do not have to worry about having duplicate names.

A registered query may contain zero or more CCL statements, and may contain one or more streams. Typically, your registered query will contain at least 1 CCL statement and at least 2 streams (at least one for input from another project and at least one for output to another project). The streams for input and output to another project will be bound to streams in the other project(s). Your registered query may also contain streams that are not bound. For example, you may have additional streams that are local streams, used only to convey data from one CCL statement in the registered query to another statement in the same registered query. The diagram below shows proper usage of the register query capability in a simple case:

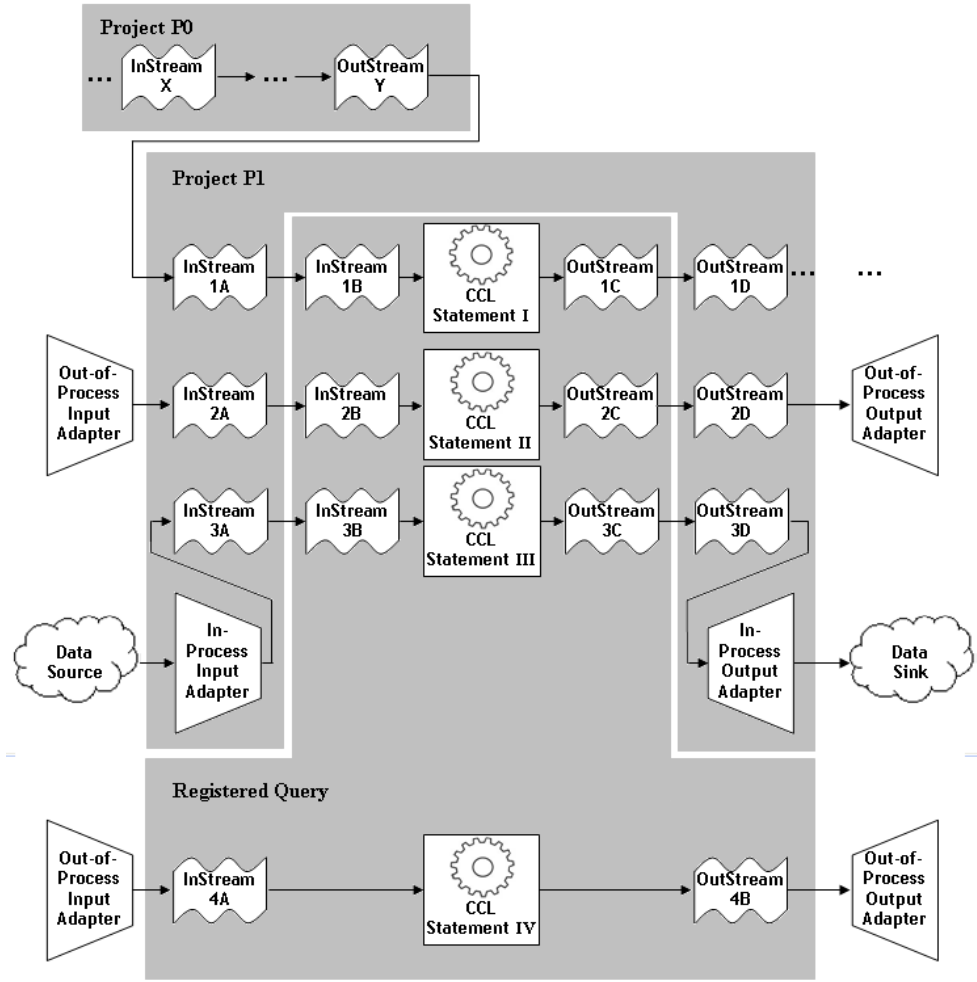


This example illustrates most of the key features and configuration elements of dynamic query registration. As you can see, we have created a project P1 that contains two streams. In this particular example, one of these streams is fed from an out-of-process input adapter, and the other stream writes to an out-of-process output adapter. When the query is registered, its CCL statement uses two streams that are bound to the streams in project P1.

Note that project P1 contains only streams.

The following is a more complex example, which shows multiple CCL statements in the registered query. Each of these statements is part of a chain with a unique combination of

streams and statements, yet the diagram shows that in each case the registered query is structured the same way - with a CCL statement(s) and input and output streams. In 3 of these examples, the input and output streams of the registered query get their data from streams in another project, while in 1 of these examples the input and output streams get their data directly from out-of-process adapters. The registered query and project P1 fit together like jigsaw puzzle pieces - the registered query "plugs in" and uses the streams that were created in Project P1.



A dynamically registered query may contain zero, or more CCL statements. In this example, the dynamically registered query contains multiple CCL statements (CCL1, CCL2, and CCL3). Each CCL statement in the registered query uses streams that are bound to a stream in the project P1. (Note that project P1 contains ONLY streams; it has no CCL statements of its own.)

The streams in project P1 may get their data from (and write their data to) different sources:

- From in-process adapters that are attached directly to the streams.
- From out-of-process adapters.
- From streams/queries in other projects (such as project P0).

Not all streams in a registered query need to be bound to streams in a project. The registered query may contain local streams, and may also contain input/output streams that are not bound. Those unbound input/output streams may be written to and read from by out-of-process adapters, and may be referenced by CCL statements in the registered query.

To use dynamically registered queries:

1. Create a project that contains only streams (no queries). Your dynamic query will bind to these streams and read and write to them. This project corresponds to project P1 in the previous diagram. When you create the streams in this project, make sure to create all of them as "input" streams.
2. If some of these streams need to have input and output adapters attached to them, then attach those adapters.
3. If some streams get their data from (or write their data to) another project(s) (for example, P0 in the diagram),create those other project(s).
4. Start execution of all of the projects, including P1. You must start executing the project(s) before you register the query(s). (When you register a query and bind its streams to the pre-existing streams, the pre-existing streams must already be running the binding operation to look up their schema information and apply that schema information to the streams that will be bound in the registered query.)
5. If some of the streams in project P1 (or other projects, for example, P0) reads from or write to out-of-process adapters, start those adapters. (We suggest starting the output adapters first, then the input adapters, so that no data is processed before the output adapters are ready for it.)
6. Run the program that dynamically registers the queries and binds to the streams in project P1.

   Why do we create all the streams in project P1 as input streams? In diagram2, it's easy to see why streams InStream1 and Instream2 are input streams - they are fed by input adapters (in-process or out-of-process). InStream3 is an input stream from the perspective of Project P1 - it receives input from another project (P0). All streams that receive input from outside their own project must be created as input streams, even if they do output as well as input.

   Please follow the guidelines listed below. Note that currently the product does not enforce all of the required restrictions by giving error or warning messages.

Stream Binding Runtime Configuration

Use macros provided by Sybase CEP to facilitate runtime configuration.

Use the macro $RuntimeFolder to path to runtime configuration files. The value of this macro is a directory called "runtime" under the directory defined by the "StoragePath" preference

under the C8/Server/Common section in the server configuration file. Stream binding runtime configuration can be set in three places:

- The project binding tab in Studio.
- The c8_compiler command line binary.
- The SDK.

In all of these cases, users are able to set the binding as "file:///absolute/path/to/fileName" or "file://$RuntimeFolder/path/to/fileName". When specifying an absolute path on windows, ensure that the format does not contain any colons. Having a colon causes a runtime error. The contents of the file must contain a valid URI on the first line without any leading or trailing spaces and should be of the form "ccl://hostname:port/Stream/workspaceName/projectName/streamName".

Use the $WorkspaceName and $ProjectName macros when specifying the location of the file. The $WorkspaceName and $ProjectName macros provide the workspace name and project name respectively.

Client-side vs. Server-side Compilation

A comparison of Client-side and Server-side Compilation functionality.

You can compile a CCL project in either the server machine or the client machine. Client-side compilation is preferred if the server is heavily loaded and if registering a query is done frequently. Server-side compilation is preferred if the client nodes are intended to be "lightweight", using minimal disk, memory, or CPU resources.

Server-side compilation has some security restrictions that client-side compilation does not. Specifically, server-side compilation is not allowed to access the hard disk drive of the server machine; the query that you want to compile must be self-contained. A query you compile on the server *cannot* do the following:

- Reference a schema that is stored in a file. The query must define all the schemas that it needs by using CREATE SCHEMA statements.
- Refer to a separate CCL module file.
- Use an adapter that was specified through Studio's GUI. You may attach an adapter only by using the ATTACH ADAPTER statement.

The table below summarizes most of the advantages of each approach:

Client-Side Compilation	Server-Side Compilation
Reduces the workload on the server, which may be important if the server is heavily loaded and/or query registration is frequent.	Reduces the workload on the client, which may be important if the client is has minimal resources. For example, CPU, memory.

	Allows light-weight clients that do not need to have the Sybase CEP Engine installed. This saves disk space and administrative work on the client, and also prevents client machines from having different versions of the Sybase CEP Engine than the server has.
Allows more than one user to compile a project at a time.	
Allows the client to access schemas that are defined in separate files.	
Allows the client to access modules that are defined in separate files.	
Allows the client to use adapters that were attached via any means other than an ATTACH ADAPTER statement.	

The specific function/method calls that allow server-side compilation or client-side compilation are documented in the chapters that describe each Sybase CEP SDK.

Guidelines

A description of rules and guidelines for using the Register Query feature.

Required

- The dynamic query must be registered after you start running the project that contains the streams that the dynamic queries bind to.
- In the project that contains the streams that the dynamically registered query will bind to, all streams that are bound to by the dynamically-registered queries must be created as "input", even if the dynamic query writes to them rather than reads from them.
- In the project that contains the streams that the dynamically registered query binds to, there should be only streams, no CCL statements.
- When you register a dynamic query, you must specify a name for each query and the name must be unique among the dynamically loaded queries in this workspace. Without this name, you will not be able to specify which query to stop later.
- The CCL implicit schema feature must be used for all local streams (if any streams are local).
- The workspace that will hold the registered query must already exist before you register the query.

Recommended

- All of the streams that a particular dynamic query binds to should be in the same project (P1 in the example above).
- The following rule applies only if all the streams in the registered query are bound streams: Before you start running any dynamic queries, you must have at least one input stream and at least one output stream. This is not required as a condition of registering the query;

however, if you do not have at least one input stream and at least one output stream in the query, the query will not be able to do anything useful.

Binding a Registered Query's Stream to an Existing Stream

Use the Sybase CEP binding mechanism to connect input and output streams of registered queries to the streams in already-existing projects. Sybase CEP uses the registerQuery() method to compile and register a query in one step.

Any CCL statement may be in the registered query. Binding streams in a query is optional.

When registering the query, provide information that binds (associates) the stream name (the name used in the CCL statement) with the URI of the corresponding stream. You must know the URI of the stream that you want to bind to.

The details are explained later in the chapters for each SDK that supports dynamic registration of queries.

Registering Streams When Registering A Query

You must create input and output streams when you register a query. Either bind those streams to existing streams in an existing project, or leave them unbound and instead use an out-of-process adapter to write data to or read data from those streams.

You must specify the schema for these streams. The individual SDK chapters explain how to specify the schema of a stream when you call `registerQuery()`.

In addition to creating input and output streams, your registered query may also create local streams. If you use local streams, do not specify a schema for them. The local stream's schemas will be deduced from the results of the CCL statements in the registered query, so you must write the CCL statements that reference those streams in a way that makes implicit schema determination possible.

Streams in registered queries are populated in one of the following ways:

- Local streams are populated via the CCL statements in the dynamic query;
- Input and output streams may be populated either:
  Via the streams to which they are bound; or,
  Via an out-of-process adapter, which will write to the stream's URI.
  To ensure that your out-of-process adapter can find the stream that you created, both the adapter and the `registerQuery()` code should follow the same rules for naming the stream. The stream name should look like the following:

  ```
  ccl://server:port/Stream/Workspace/QueryName/StreamName
  ```

  where
  Workspace is the name of the workspace that you specify when you call the `registerquery()` function/method.
  QueryName is the name of the query that you specify when you call r`registerQuery()`. This name must be unique within the workspace.

StreamName is the name of this specific stream. This stream name must be unique within the query.

Stopping A Query

Each Sybase CEP SDK that supports query registration also contains a "stopQuery()" function or method.

The exact name of the function or method depends on the SDK. To stop the registered query, call the stopQuery() function and pass it the name of the query, which you specified when you registered the query. This will stop the query and remove it from the server. (The c8_client "stop" command will do the same.)

Troubleshooting the Perl SDK

This section provides troubleshooting tips that are specific to the Perl SDK.

If you get an error similar to the following:

```
Can't locate SybaseC8/Publisher.pm in @INC (@INC contains: /usr/lib/
perl5/5.8/cygwin /usr/lib/perl5/5.8 /usr/lib/perl5/site_perl/5.8/
cygwin /usr/lib/perl5/site_perl/5.8 /usr/lib/perl5/site_perl/5.8/
cygwin /usr/lib/perl5/site_perl/5.8 /usr/lib/per l5/vendor_perl/5.8/
cygwin /usr/lib/perl5/vendor_perl/5.8 /usr/lib/perl5/vendor_perl/
5.8/cygwin /usr/lib/perl5/vendor_perl/5.8 .) at examples/
sybasec8_publish.pl line 10. . . .
```

The most likely cause is that you have not set your Perl include path to include the Sybase CEP directory that includes the Perl modules. To configure your environment properly, see *Prerequisites* on page 303.

If you get an error similar to the following:

```
Can't locate URI/URL.pm in . . .
```

The most likely cause is that you have not set your path to include the appropriate CPAN modules. For a list of these modules, see *Specifying the Path to the Library Files* on page 309. If you have not acquired these modules, you may need to do so. If you have acquired them, you need to set your perl include path to include these modules.

Additional troubleshooting tips are in *Troubleshooting* on page 351

Monitoring Servers and Projects

This section describes how you can monitor servers and the projects that are running on those servers.

Server status information

The Sybase CEP Server provides extensive status information to help you monitor servers and projects, including information about CPU utilization by a particular server, the number of messages received by a particular CCL query, managers, containers, workspaces, projects, and queries.

Sybase CEP provides status information about the following types of entities:

- Manager.
- Container.
- Workspace.
- Program (also called "project").
- Query.

While other types of status messages exist, these are of most use to the typical user.

All status information is collected and made available by the manager; you will not receive status information directly from individual containers. If the manager is down (or, in a High Availability system, if all the managers are down), you will not be able to read status information.

Status information can be retrieved in different ways:

- From Status streams, which are similar to other streams.
- From the function/method calls provided in particular Sybase CEP SDKs, such as the C/C++ SDK and Java SDK.
- From server plugins.

Each of these is described in more detail below.

Status Streams

Each manager provides one status stream. The stream may contain status messages about different entities, such as, managers, containers, workspaces, projects, and so on. The messages are published at regular intervals.

Other information is published only when something changes state. For example, a message about a container being added is sent only when a container is actually added. As in any stream, all messages in the status stream have the same schema. The schema includes the following fields:

SourceTimestamp	The SourceTimestamp indicates when the event occurred. For example, in a Status message, SourceTimestamp will be the time when the last Status change took place. Therefore, the SourceTimestamp in a status message will frequently be lower than (earlier than) the row timestamp in the same message. (For information about the row timestamp, see the Sybase CEP *Programmer's Guide* .) The row timestamp in the message is simply when the message was sent, not when the event described in the message actually occurred. For data sources that send the message as soon as the event occurs, or for messages containing info that changes frequently (such as the Message Count messages), the SourceTimestamp and row timestamp are usually very close.
ObjectID	The object identifier. For example, if we are requesting information about a project, the object ID will uniquely identify which project we want information about. The objectID is often in the form of a "path", in other words, a string that contains a series of names separated by slashes. For example, the objectID of a query is in the following form: `<WorkspaceName>/<FullCclPath>/<StatementNumber>`
ObjectID2	The object identifier of the second object (if applicable). For example, if we are requesting information about latency between 2 streams, then the first and second ObjectIDs are the URIs of the two streams, each of which will (like any stream URI) be in a form similar to the following: `ccl://proton:6789/Stream/Default/RFID_Adapter-ReadPallet/Out`
MessageGroup and MessageName	The combination of MessageGroup and MessageName specify the "topic" of the message. For example, if the object ID is the ID of a container, then the combination of MessageGroup "ContainerInfo" and MessageName "HeartbeatPeriod" indicate that the message is about the frequency with which the container sends heartbeats to the manager. For more detail about the MessageGroups and MessageNames, see *Status Information* on page 519.
Value	This is the specific value of the message, which is always in the form of a string. For example, if the MessageName is "HeartbeatPeriod", then the value might be "5" to indicate 5 seconds.

As you can see, with only 6 fields (plus the row timestamp), the structure of status messages is simple. However, since there are dozens of different types of messages (in other words, dozens of different message names), the code that you write to receive and use status messages must "interpret" the messages appropriately. For example, if you want to perform mathematical operations on the number of messages received so far by a particular query, you will need to convert the message value string to a numeric value. For information about currently supported MessageNames that you may see in status messages, see *Status Information* on page 519.

To view a status stream, do the following:

1. Start Studio.
2. Click on the "Debug" menu.
3. Select the "View stream..." option.
4. When the dialog box is displayed, enter one of the following URIs:

```
ccl://managerhost:port/Status/Workspace/WorkspaceName
    ccl://managerhost:port/Status/Service/Manager
```

   For example, to view the status stream for the workspace named "Default" when the manager is running on the machine named "penguin1", use the URI.

```
ccl://penguin1:6789/Status/Workspace/Default
```

Status APIs

Many of the Sybase CEP SDKs, including C/C++, Java 1.5, and .NET3, enable you to get status information by calling particular functions/methods.

To get information about a particular object (for example, a container or a query), you must provide two pieces of information:

- The unique ID of the object.
- An indicator of whatinformation you want about that object.

To specify the ID of the object, use the objectID. The objectID was described briefly above; below outlines how to compose the object ID for each type of object:

Object Type	ObjectID
Manager	Manager URI.
Container	Container URI.
Workspace	Workspace name.
Project	The ObjectID is the full path to the given project in the form: <Workspace-Name>/<ProjectName>. ProjectName is the project's load name, or the name that is displayed in Sybase CEP Studio's Explorer View. This is usually, but not always, the same as the name of the project file without the ".ccp" extension.
Query	The ObjectID is the full path to the given Query in the form: <Workspace-Name>/<FullCclPath>/<StatementNumber> Where <FullCclPath> is the "path" of the loaded module as you see it in Studio's Explorer View (for example, `Project1/SubModule1/SubModule2`) and <Statement-Number> is the sequential number of the CCL statement in the module.

To specify what information you want, specify both the Message Group and the Message Name. A complete list of Message Groups and Message Names is available in *Status Information* on page 519.

In general, the Message Name indicates the specific information that you want. For example, the percentage of CPU time utilized by an object (such as a container). Because some types of objects have similar information (for example, both modules and individual queries have status information about the number of "InputMessages"), you must specify a MessageGroup which combines with the Message Name to indicate what information you want. To get a specific value (for example, the number of input messages received by a particular query), you must specify all three of the objectID, MessageGroup, and MessageName.

You will first receive a status object that contains information about a Manager, a Workspace, or a Project (a "top module"). Once you have that object, you can call a function that queries that object for more specific information, such as the number of rows received by a particular query within the project. Note that a single status object may contain many status messages. For example, if a status object for a top query module ("project") has ten different message types (MessageNames), then when you retrieve a status object for a particular query you will be able to extract ten different messages from it. In pseudo-code, this would look similar to the following:

```
char messageGroup[] = "CclQueryInfo";
char messageName[] = "InputMessages";
// objectID = workspace name + full CCL Path + query number.
char objectID[] = "AccountingWorkspace/fullCCLPath/2";
C8Message *statusMsg;
char *bufferPtr = NULL;
// Get the status object for a project ("top module").
statusObject = C8StatusObjectForTopModule(managerUri,
           workspaceName, moduleName);
numMsgs = C8StatusObjectGetMessagesNumber(statusObject);
for (i = 0; i < numMsgs; i++) {
 statusMsg = GetStatusObjectMessageAtPos(i, messageGroup,
             objectID, messageName);
 C8MessageColumnGetAsStringByName(statusMsg, "Value",
             &bufferPtr);
 msgsRcvd = stringToInteger(&bufferPtr);
 }
```

More specific information about the function/method calls is provided in the chapter for each SDK that supports status information.

User Authentication from Inside SDKs

The Enterprise Edition of Sybase CEP supports a user authentication feature. This enables an administrator to limit access to the Sybase CEP Server, or to certain objects within the server.

Note: This section applies only to the Enterprise Edition of . If you are using the Professional Edition, you may skip this section.

If user authentication is enabled, then client applications, such as out-of-process adapters and programs that register queries, must authenticate themselves with the server by providing a username and password when calling certain API functions.

The following operations require authentication:

- Connecting to a stream (as a subscribing or publisher).
- Retrieving a stream schema.
- Resolving a URI.
- Starting or stopping a Sybase CEP project.
- Getting the status of a server.
- Getting the status of a workspace.
- Getting the status of a project (also called an "application").

The Sybase CEP C/C++, Java 1.5, and .NET3 SDKs support user authentication.

In general, you will follow these steps to authenticate:

1. Create or obtain an empty C8UserCredentials object.
2. Fill in the C8UserCredentials object with the user name and password.
3. Perform the desired operations (for example, connect to a stream).
4. Destroy the credentials.

   This process assumes that you have already server has already been configured to request user authentication.

   Depending on which user authentication rules have been established, a single program might need to perform authentication for some operations and not for others. For example, if any user is allowed to subscribe to a stream, but only certain users are allowed to start and stop the project that contains that stream, then a client program that starts and stops the project and also subscribes to the stream will follow the user authentication rules for starting the stream but not for subscribing to it.

   For more information about user authentication, including how to turn the feature on and how to configure it, see the *Sybase CEP Installation Guide* .

   If you will be creating your own plugin to do custom authentication, see *How to Implement Manager HA with the Sybase CEP Generic Plugin* on page 336.

User-Defined Functions and Plugins

You can write User-Defined Functions (UDFs) and server plugins to extend the server of the .

Note: In this manual, when we refer to User-Defined Functions, we are referring to C/C++ UDFs unless stated otherwise. Starting with version 5.1.0 of the Sybase CEP Server, you may also write user-defined functions by using CCL's CREATE FUNCTION statement. See the for information about CCL's CREATE FUNCTION statement.

This chapter also provides a conceptual overview of plugins. A detailed explanation of how to write a plugin is in *Server Plugins* on page 327.

User-Defined Functions

Explains the concept of user-defined functions and outlines their requirements.

Functionality

To create a User-Defined Function (UDF), you must write your own code, dynamically link that code into the server, and then call that code from within a CCL statement.

For example, if you wanted to calculate the monthly mortgage payment on a loan, you could write a UDF named "MonthlyMortgagePayment()" and call it as shown below:

```
INSERT INTO ...
SELECT MonthlyMortgagePayment(loanAmt, interestRate, NumOfMonths)
...
```

When the user-defined function is called from a CCL query, the Sybase CEP Server will call the function with the appropriate data. In the example above, the user's function will be passed the values that are in the loanAmt, interestRate, and numOfMonths columns of the current row. The parameter(s) passed to the function must be in the form of a proper CCL expression, which may be a column name, a literal expression, or a more complex expression. For example:

```
INSERT INTO OutputStream
SELECT *
FROM InputStream
WHERE f1(InputStream.MyIntCol, InputStream.col2 / 100,
'A Literal', f2(f3(x))) > 1000;
```

A UDF can appear in most parts of a CCL statement where a general expression can appear.

A UDF can appear more than once in the same statement. For example:

```
INSERT INTO stream_Out
SELECT
MY_UDF(my_col)
 AS f_Current,

MY_UDF(my_col)
 - PREV(my_col) AS f_Delta
FROM stream_In;
```

UDFs may be scalar functions or aggregate functions. Scalar functions return a value based on a single set of inputs (for example, one or more values from a single row, possibly with other expressions that are independent of any row). The built-in SQRT() function is a typical scalar function. Aggregate functions return a value based on 0 or more rows. The built-in SUM() and AVG() functions are typical aggregate functions.

For each row, if the same UDF is called with the same parameter value(s), the server may either execute the function once for each time the function is called from within the statement, or the server may call the function fewer times and cache the results. The server may then use the cached results some of the times when the function would have been called.

- For aggregate functions, the server will call the function once for each time that the function exists in the statement.
- For non-aggregate functions, the number of times that the function is called (with the same parameter value) is determined internally by the server and is not predictable or controllable by the user.

If you are writing your own UDF, in almost all cases the function should be idempotent, in other words, it should return the same value regardless of how many times it is called within the same statement. Only in very rare cases, such as the built-in NEXTVAL() function, should a function deliberately return a different value for each call with the same value inside the same statement.

A UDF can contain almost any code that you can write in any of the languages for which Sybase CEP provides an SDK that supports UDFs.

UDFs allow you to provide for your own specialized needs. The range of capability that you may add is almost unlimited. The UDF may even go "outside" the environment of the Sybase CEP Server in which it is running. For example, you might write a function that would query a high-precision pi() value from a university mainframe using SOAP (Simple Object Access Protocol).

Requirements

This section outlines the requirements for various function types.

UDF requirements

Each user-defined function (UDF) must meet several requirements.

The server must be able to find the library that contains your UDF. To facilitate this, load the library into memory, call your UDF, and pass the appropriate parameters to the UDF. Each UDF must meet several requirements:

- Before the server can call a function, the server must be told the name of the library, the names of the UDFs in that library, and the data types of the parameters passed to each function. This information must be stored in an XML file that the server reads when it starts up. The format of this file is described in *UDFs: XML Signatures* on page 242.
- The UDF must declare appropriate C-language data types that match the CCL data types of the parameters. For example, if the server calls the UDF with a value of type TIMESTAMP, the corresponding C-language variable must be of type C8Timestamp.
- The UDF must be written in C or C++. The externally visible function names must use C naming conventions. Improper C++ naming will make the code inaccessible.
- The UDF implementation must be compiled into a shared library (also called a Dynamic Link Library) for its specific platform. That library may contain one or more user-defined functions.

Some of these requirements may be modified in future versions of the product as more Sybase CEP SDKs are enhanced to support UDFs.

To write a UDF, you must create the following files:

- A C-language file that contains your UDF and code to pack and unpack values that are stored in a context parameter. (This is described in more detail later.)
- An XML file with the library name, function name, and parameter types.

Each of these files is described later in this document.

User-Defined Aggregate Functions

To write an aggregate function, you must have a place to store your data between invocations. For example, if you write your own "SUM()" function, you must store the subtotal from the previous calls to the function and then add the new value from the current call to the function.

Since an aggregate function may be called by many queries at overlapping times, you cannot use local storage space to store values such as subtotals. If you store the subtotal in a static variable declared in the routine, then each different call from different queries would update the same variable, and the result would be meaningless. A similar problem occurs if you try to use global variables. Non-static local variables (in other words, stack-based variables) can't be used because the value may be overwritten between calls to the function.

Your aggregate function must work with the server to store data in a place that will persist after the UDF returns from its call. Sybase CEP provides a pair of functions that enable you to store and retrieve data. These functions are documented in the relevant SDK chapters.

The bytes stored and retrieved are specific to a particular occurrence of a particular function in a particular CCL statement. If your query module has the following statements:

```
...
SELECT aggregate_foo(col1)
FROM InStream1
...
SELECT aggregate_foo(col2)
FROM InStream2
```

then each "aggregate_foo" will have a unique internal identifier that the server uses as part of the context variable passed to your function. The context variable then allows the server to know which aggregate_foo's bytes to return. This is explained in more detail in the relevant SDK chapters.

There is a second hurdle that an aggregate function must overcome. An aggregate function operates on a window (for example, the window created by a "KEEP 3 ROWS" or "KEEP 10 MINUTES" clause). Such a window has rows entering and leaving. If the window is defined by a KEEP 3 ROWS clause, for example, then each time that a new row is received, the new row will displace the oldest row. However, if you are writing a SUM() function, and you only want the sum of the three most recent records, then each time that a new record arrives, you must not only increase the sum based on the value from the most recent record, but you must also decrease the sum based on the value from the record that was displaced from the window.

To ensure that your aggregate UDF takes into account the displaced record as well as the new record, your aggregate UDF will usually be called twice each time that a new record arrives. Your aggregate UDF will be called once with the value of the new record and once with the value of the displaced record. Inside your aggregate UDF, you will distinguish between the new and displaced values by calling a function named C8IsPositiveMessage() or C8IsNegativeMessage(). When the function is called with the newly-added value, then C8PositiveMessage() will return C8_TRUE, and C8NegativeMessage() will return C8_FALSE. When the function was called with the displaced value, then C8PositiveMessage() will return C8_FALSE and C8NegativeMessage() will return C8_TRUE.

XML Signatures

Sybase CEP software requires the description of the UDF to be in an XML file, which is put in the plugins directory of both the server and Studio.

The file name should have the extension ".udf" so the server and Studio can find the file. For each UDF, the file lists:

- The name of the function.
- The name of the library file that contains the function.
- The data types of each of the parameters to the function.
- The data type of the return value of the function.

The Functions element of the XML file may describe an arbitrary number of functions. This allows for the grouping of related functions and/or library modules, and there may be an

arbitrary number of function modules in a library. In the XML file, each function should be described inside its own "<Function> ... </Function>" element. For example:

```
<Functions>
<Function Name="MyFunc"
         InitFunctionName="MyInit"
         ShutdownFunctionName="MyShutdown"
         Library="c8_udf_lib"
         CclName="MyFunc"
         IsAggregator="false">
 ...
 <Input>
    <Parameter Name="var1"  Type="C8Float"/>
    ...
 </Input>
 <Output>
    <Parameter Type="C8Long"/>
 </Output>
</Function>
<Function Name="bar" ...>
...
</Function>
</Functions>
```

The initial function element is always named "Function" and has six attributes:

- The Name attribute provides the name of the function that will be called from the user-designated Library. This name is case-sensitive.
- The optional InitFunctionName attribute identifies a function that will be called on initialization.
- The optional ShutdownFunctionName attribute identifies a function that will be called on shutdown.
- The Library attribute is the name of the .dll containing the function. For UNIX-like operating systems, this will be a .so library, and should not be prefaced with "lib". Note that the extension .dll or .so is NOT included as part of the library name in the xml file. You may use the same XML file for both UNIX-like operating systems and Microsoft Windows environments without changing the library name.
- The CclName is the name that will be used in CCL queries (for example, "select FOO() ... from ..."). The CclName is case insensitive. The CCL name is usually the same as the function name, but this is not required.
- The IsAggregator attribute should be true for aggregate functions and false for other functions.

The Input element inside each Function element must contain a list of all input parameters. Only one Input element is permitted per function.

Each Parameter element inside the Input element includes two attributes: the parameter "Name" and "Type". The Name is arbitrary, but for clarity it should reflect the parameter usage. The Name attribute is used for error reporting if there is a type mismatch for any parameters at runtime.

The "Type" parameter specifies the data type of the input parameter. If you are writing your UDF in C/C++, the types available map directly into C base types via these typedefs:

```
typedef int C8Bool;        /* CCL "BOOLEAN" type */
typedef int32 C8Int;       /* CCL "INTEGER" type */
typedef double C8Float;    /* CCL "FLOAT" type. Note that
                        * CCL FLOAT is NOT C "float";
                        * it's C "double".
                        */
typedef char* C8CharPtr;   /* CCL "STRING" type */
typedef char C8Char;
typedef int64 C8Long;      /* CCL "LONG" type */
typedef int64 C8Timestamp; /* CCL "TIMESTAMP" type */
typedef int64 C8Interval;  /* CCL "INTERVAL type */
typedef void *C8BlobPtr;   /* CCL "BLOB" type */
typedef void C8Blob;       /* CCL "BLOB" type */
```

For example, if the first input parameter has a CCL data type of FLOAT, then you would specify C8Float in the .udf file. Only the above typedef names are supported in the "Type" attribute. Throughout the C code for your UDF, you should use the typedef'd names (for example, "C8Float") rather than the underlying C types (for example, "double"). The Sybase CEP software uses strong type checking to help ensure correctness of data at runtime.

The "output" element of the sample XML file contains only a single "Parameter" field and describes the function output in the same manner as input parameters. The "Name" attribute is optional. It is not currently possible to return more than a single value from a user defined function.

Sybase CEP SDKs that Support UDFs

Currently, UDFs are supported by the Sybase CEP C/C++ SDK.

For much more detail, see *User-Defined Functions* on page 234.

Plug-Ins

This section explains plug-ins and their functions.

Plug-Ins: Overview

Plugins are executable code that expand the functionality of the server.

Examples of plugins include the user authentication plugin (see *The User Authentication Plugins* on page 337) and the server status plugin, which you can use to help implement customization of existing functionality, such as High Availability (see *How to Implement Manager HA with the Generic Plugin* on page 336).

Unlike UDFs, plugins are called by the server rather than called from yourCCL statements.

Every plugin must have at least three functions:

- An "initialize" function.
- An "execute" function.
- A "shutdown" function.

The initialize function performs any initial setup that is required. This may involve allocating memory or looking up information outside the server. If no setup is required you must still have an initialize function; however, you may leave it empty.

The execute function is called each time there is a reason for the plugin to do something. For example, in a user authentication plugin, the function would be called each time the server needs to verify that a user's ID and password are valid. In a Manager HA plugin, the execute function might be called each time a manager fails and needs to be replaced with a standby manager.

The shutdown function allows the plugin to do any necessary cleanup, such as deallocating memory. If no cleanup code is required, you must still have a shutdown function even if it is empty.

To call these functions, the server must know the names of the functions and the name of the linkable library file (typically a .dll or .so file) that contains the functions. The name of this file, as well as the names of the initialize(), execute(), and shutdown() functions, must be specified in the server configuration file (c8-server.conf). An example of part of a configuration file is shown below.

```
<section name="ManagerFailoverDDNSPlugin">
<preference name="LibraryName"
          value="c8_server_plugins_lib"/>
<preference name="InitializeFunction"
          value="c8_command_line_plugin_initialize"/>
<preference name="ExecuteFunction"
          value="c8_command_line_plugin_execute"/>
<preference name="ShutdownFunction"
          value="c8_command_line_plugin_shutdown"/>
...
</section>
```

The plugin may be very small (just big enough to run an external program and pass it appropriate parameters), or the plugin may be large if that is necessary to handle the work required.

For some plugins, the timing of the call to the plugin's execute() function may be determined by an event in a stream. For example, when an HA manager dies, an event is published to a status stream. The server monitors this stream and calls an HA-related plugin when the stream contains a message indicating that the HA manager died. For other plugins, such as the user authentication plugin, the activity (such as a user trying to access a restricted resource) may not show up as an event in a stream.

If the plugin is invoked based upon the arrival of a message in a status stream, then you will also need to specify which message should cause the server to invoke the plugin. An example of a c8-server.conf file section specifying the message is below:

```
<section name="ManagerFailoverDDNSPlugin">
...
<preference name="MessageGroup"
          value="ManagerInfo"/>
<preference name="MessageName"
          value="ManagerHAPromotedToPrimary"/>
...
</section>
```

For a more complete example and lists of valid MessageGroups and MessageNames, see Message-Driven Plugins shows

Sybase CEP SDKs that Support Plugins

Currently, plugins are supported by the Sybase CEP C/C++ SDK.

For much more detail, see *User-Defined Functions* on page 234.

Other SDKs may support plugins in the future.

Remote Procedure Calls, Database Queries, and Public Windows

An overview of how to integrate your Sybase CEP Server with remote servers.

Sybase CEP Server can retrieve data from input adapters. Sybase CEP Server can also retrievedata from, or invoke processing on, Remote Procedure Call (RPC) servers and database servers (for example, Oracle, MySQL, SQLServer, and so on).

You can write a program that reads data from a public window (a window that exists inside Sybase CEP Server but can be accessed from outside Sybase CEP Server as though the window were a database table.)

Please note that this chapter does not completely document this topic. For more information, see:

- *Sybase CEP Installation Guide* : This explains how to configure your c8-services.xml file to tell your Sybase CEP Server how to connect to remote servers.
- *Sybase CEP CCL Reference Guide* : This documents the CCL language statements and clauses that are capable of accessing remote servers. The Sybase CEP *CCL Reference Guide* also documents the CREATE PUBLIC WINDOW statement (which is used to create a public window that can be accessed externally) and Sybase CEP's SQL (which can be used to query a public window.)

Remote Procedures

This section describes how to call a remote procedure from a CCL statement.

Calling a Remote Procedure from a CCL Statement

Sybase CEP's RPC feature enables CCL statements to execute a function or procedure that is running outside the Sybase CEP Server process, either on the same computer as Sybase CEP Server, or on a different computer.

Note: This guide uses the term RPC to refer both to procedures (which do not return a value) and to functions (which return one or more values).

RPCs vs. UDFs

An overview of UDF and RPC features, and in what scenarios they work best.

RPC Features:

- The RPC code exists in a form that cannot be made into a .so or .dll file that Sybase CEP Server file can use. Reasons for this include:

- The code is supplied by a 3rd party and is supplied only as an executable library, not a source code.
- The code is not written in a language that Sybase CEP SDKs support for UDFs.
- The RPC must be in a separate process so that if it crashes it will not affect the Sybase CEP Server.
- The RPC is resource intensive. Run it on a separate machine from Sybase CEP Server to spread the workload.

UDF Features:

- UDF calls are usually much faster than RPC calls. This is true for both CCL UDFs and C/C++ UDFs.
- It is often much easier to write a UDF in CCL than to write an RPC and configure the server to call that RPC.

What Is a Remote Procedure Call?

RPCs are similar to User-Defined Functions (UDFs). Both UDFs and RPCs enable you to extend Sybase CEP Server by writing custom code. A UDF runs as part of the server process, while an RPC runs as part of a separate process.

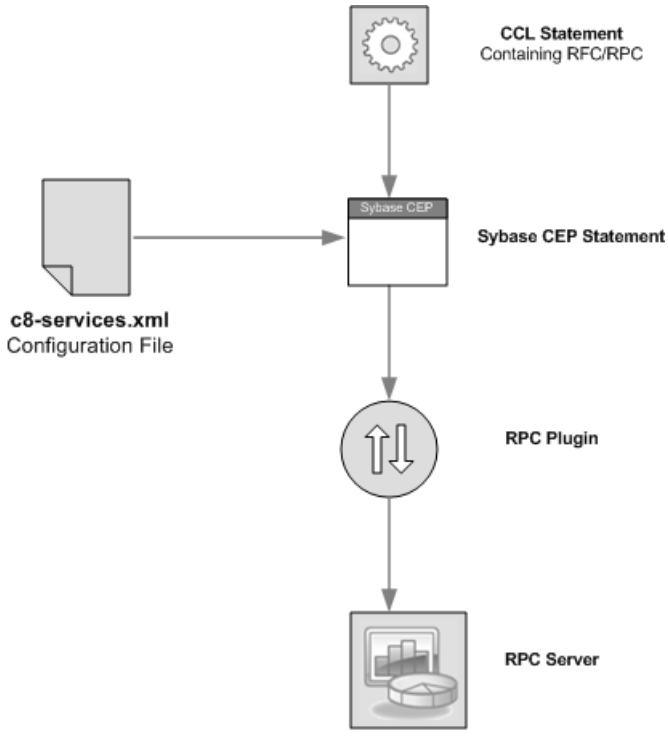
The Components

This section describes the software components required for Sybase CEP RPCs and how they interact with one another.

Remote Procedure Call Components

A description of the multi-step RPC procedure and the software components involved.

As compared to a CCL statement, an RPC is a simple operation. However, each RPC call involves multiple steps. The following diagram shows the components involved and how they are related:



RPC functionality requires the following software components:

- A Sybase CEP Server.
- A project (query module) that includes a CCL Remote Procedure Statement or remote subquery that calls the remote procedure. For more information about this CCL statement and subquery please refer to the Sybase CEP CCL Reference.
- An RPC plugin that interfaces between and the RPC server. When the server wants to call a remote procedure (or function), the server calls a function inside this RPC plugin, and the RPC plugin then calls the remote procedure and returns any values to .

  The RPC plugin must know how to call the remote procedure. For example, the RPC plugin may know the HTTP URI of the remote procedure and know how to execute an HTTP POST operation to invoke that remote procedure. Sybase CEP Serveritself does not need to know how to communicate with every remote procedure that is written; the plugin handles the communications.

  Complile the RPC plugin as a dynamically linkable library (.dll file) or shared object library (.so file) so that Sybase CEP Servercan call it.
- A section of the c8-services.xml file that describes the RPC plugin and enables the to call it. You must describe each RPC plugin in a separate section of the c8-services.xml file.

  For each remotely callable procedure, Sybase CEP Server must know the name of the library file that contains the corresponding RPC plugin.

- An externally executable program that includes a procedure or function that you can call remotely. The program must be capable of receiving and responding to requests from external applications. This manual refers to such programs as RPC servers. The RPC server may run on the same computer as the or on a different computer.

These components work together as follows:

1. The CCL Remote Procedure Statement or remote subquery invokes the RPC plugin by specifying the plugin's unique service name (as identified in the c8-services.xml file) and may also pass values to the plugin, which are then passed on to the RPC server.
2. The uses the service name and the information defined for the service in the c8-services.xml file to determine which RPC plugin to use and which function(s) in that plugin to call. The Sybase CEP Servercan also read some plugin configuration parameters (defined in the c8-services.xml file service entry) and make these available to the plugin, which can read them by calling appropriate function(s) in the API.
3. The RPC plugin communicates with the RPC server and invokes the remotely callable procedure. When appropriate, the plugin passes remote procedure parameter values from the CCL query to that RPC and/or receives values back from the RPC and makes them available to the CCL query.

Remote Procedure Call CCL Statement

A description of the syntax rules for the RPC CCL statement.

The CCL statement may be either an EXECUTE REMOTE PROCEDURE statement or may be a subquery that makes a remote query call. Below is a pseudocode example of each:

```
EXECUTE REMOTE PROCEDURE 'MyRPC1'
SELECT ...
FROM StreamIn1;
INSERT INTO StreamOut1 (StringCol1)
SELECT RFCOutputAlias.StringCol1
FROM
  (REMOTE QUERY "MyRFC1" SCHEMA StringOnlySchema
      (
      StreamIn1.StringCol1 AS Name
      )
  ) AS RFCOutputAlias,
  StreamIn1 KEEP 5 ROWS;
```

The RPC is called each time a new row arrives in any of the streams involved in the CCL statement.

Note that, as with database subqueries, Sybase CEP Server has no knowledge of when information changes on the remote system. The RPC cannot notify Sybase CEP Server that data has changed; the server does not see updated information until a row arrives in one of the query's streams and the server then calls the RPC.

For more information about the syntax of these CCL statements and clauses, see the Sybase CEP *CCL Reference Guide* .

Remote Procedure Call Component Services file

A description of how to modify the c8-services.xml file in order to use a RPC Component service file in CCL queries.

The c8-services.xml file must contain a section for every remote service before you can use the service in a CCL query. Each service has a service name (used in the CCL statement) and information about the RPC to be called.

Your specific service entry depends on the RPC plugin and remote procedure you are using. See the *Sybase CEP Installation Guide* or *HTTP and SOAP Plugin Preferences* on page 361 for more details.

The Sybase CEP Server

A description of the Sybase CEP Server process for executing CCL statements that contain RPCs.

Sybase CEP Server executes the CCL statement that contains the RPC and then invokes the appropriate function(s) in the plugin to send data to (and receive data from) the remote procedure.

To avoid the overhead of an RPC call, the server may cache information retrieved by the plugin and read data from that cache rather than re-executing the remote procedure call. For information about setting cache expiration times, see the *Sybase CEP Installation Guide* .

The RPC Plugin

A description of the RPC plugins provided by and how they operate.

RPC plugins are compiled .dll files (on Microsoft Windows) or .so files (on UNIX-like operating systems). These libraries are called by the Sybase CEP Server, and are able to communicate with RPC servers. The RPC plugin calls the remote procedure, typically using Simple Object Access Protocol (SOAP), HTTP, or another protocol appropriate for the specific remotely callable procedure.

The plugin receives information (such as the body of an email message, a stock symbol, or an employee ID number) from the CCL statement and passes the information to the remote procedure. The plugin also receives information from the remote procedure (when appropriate) and makes it available to the CCL statement.

The RPC plugin you are using must be designed and configured to work with the specific remote procedure you are using. The plugin must be able to contact the RPC server (for example, via an HTTP URL), must invoke the name of the desired remote procedure on that RPC server, and so on. In some cases, the appropriate plugin may be obtainable from a third-party vendor. You can also write your own RPC plugin.

provides two generic plugins that allow you to make remote procedure calls to RPC servers that handle SOAP and HTTP protocols. Each of these plugins may be used as-is or customized

for your needs. The generic SOAP plugin includes an example SOAP RPC plugin and corresponding service entry in the c8-services.xml file.

Note: Do not confuse the RPC plugins discussed in this chapter with Sybase CEP Server plugins discussed in *HTTP and SOAP Plugin Preferences* on page 361. Although there are some similarities, they are not the same.

The RPC

The remotely callable procedure runs as an independent process, not as part of Sybase CEP Server. Run the RPC server on the same computer as the Sybase CEP Server, or on a different computer.

The RPC server can serve many different clients. Furthermore, the Sybase CEP Server, plugins, and CCL queries can be clients to several different RPC servers. One RPC server could check email for viruses, while another looks up historical stock information, and a third retrieves customer information.

You or a 3rd party can write the RPC server. You can write the RPC server in any language and are not limited to the languages thatSybase CEP provides SDKs for.

Additional Information about RPCs

A description of situations where will call a remote procedure.

Sybase CEP Server calls a remote procedure when:

- A new row arrives in one of the input streams referenced by the CCL statement that contains the remote procedure call.
- The information in the cache is expired. You can set a c8-services.xml file parameter to determine how long the server retains cached responses from the remote procedure calls.

Note: Note that RPC calls are not "blocking calls" within Sybase CEP Server. See *Remote Requests, Synchronization, and Performance* on page 64 for more details.

Generic HTTP and SOAP Plugins

A description ofSybase CEP sample Generic HTTP and SOAP plugins. Use these plugins with remote procedures that are called via HTTP or SOAP protocols.

The Sybase CEP product includes two example RPC plugins:

- Use the HTTP plugin to send HTTP POST requests. The source code for this plugin is in the file `c8_rpc_http.cpp`.
- Use the SOAP plugin to send SOAP requests. The source code for this plugin is in the file `c8_rpc_soap.cpp`.

Use these plugins with remote procedures that are called via HTTP or SOAP protocols. When you specify the name of the compiled library file in the `c8-services.xml` file, specify only the base name `c8_rpc_http_soap_lib`. The server automatically appends the

appropriate filename extension (.dll or .so) for the operating system. On UNIX-like operating systems, the initial "lib" of "libc8_rpc_http_soap_lib.so" is prefixed, if necessary.

The Sybase CEP HTTP plugin enables you to pass zero or more parameters and get back zero or one parameter(s).

The Sybase CEP SOAP plugin enables you to pass zero or more parameters and get back zero or more parameter(s).

If you need to retrieve more than one parameter, use the SOAP plugin or a custom plugin.

Source code for both libraries is in the `server\sdk\c\examples` subdirectory of your Sybase CEP installation directory. Typically, this is either:

```
C:\Program Files\SybaseC8\Server\sdk\c\examples
or /home/<userID>/Sybasec8/server/sdk/c/examples
```

The HTTP plugin supports HTTP 1.0. The plugin expects exactly one column of type STRING from the CCL statement and produces exactly one STRING result in the HTTP response body. The HTTP plugin recognizes HTTP errors and returns nothing to the CCL statement if it encounters such an error.

The SOAP RPC protocol supports a subset of SOAP 1.2 (see *http://www.w3.org/TR/soap12-part1/* and *http://www.w3.org/TR/soap12-part2* for more info). The SOAP RPC plugin expects any number of input columns from the CCL statement and can produce any number of output columns. The SOAP plugin recognizes SOAP errors and returns nothing to the CCL statement if it encounters such an error. For information about the conversion of CCL datatypes into XSI types and vice versa, please see Appendix I of the Sybase CEP *CCL Reference Guide* .

The parameters expected by the HTTP and SOAP plugins are detailed in *HTTP and SOAP Plugin Preferences* on page 361.

Sybase CEP provides these two plugins in source code form. Compile the plugins before you use them, and put the compiled code into the server's "bin" directory.

Remote Procedure Call Plugin for CSV Files

Sybase CEP provides an RPC plugin that reads the contents of a file in CSV format.

Configuring the Plugin

To configure the RPC plugin, modify the related section of the file `c8-services.xml`, located in the conf subdirectory of your Sybase CEP Server installation directory. The file itself contains comments and an example configuration, and the *Sybase CEP Installation Guide* provides instructions for enabling RPCs.

Besides the parameters common to all RPC plugins, this plugin uses two additional parameters:

- BaseFolder: (Optional). When you call this plugin and specify a file, the file must be located somewhere beneath the base folder specified here. If you do not specify a value for this parameter, Sybase CEP Engine uses the value of the ReadWriteBaseFolder preference specified in `c8-server.conf` instead.
- Format: (Optional). The format of the file, in the form "TitleRow= true | false, TimestampColumn= true | false." The defaults are true and false, respectively.

Using the Plugin

When you use the plugin in a query, the filename parameter is required. The parameter specifies the path and name of the file to read. Use $BaseFolder to insert the path of the base folder or $ProjectFOlder to insert the path of the folder containing your project. By default, paths are relative to the base folder.

Be aware of the following behaviors when calling this plugin:

- The plugin reads the entire CSV file for every row that triggers the query calling the RPC.
- The resulting rows have timestamps matching the timestamp of the row that triggers the query calling the RPC. Any timestamps in the CSV file itself are ignored.

Examples

This first example shows how to read the file `myconfig.csv` (located in the same folder as the project file) at startup and populate a window with the contents:

```
CREATE LOCAL STREAM StartupTrigger SCHEMA (X INTEGER);
INSERT INTO StartupTrigger
VALUES (1)
OUTPUT AT STARTUP;
INSERT INTO MyConfigWindow
SELECT CsvFile.*
FROM
   StartupTrigger,
   (REMOTE QUERY "ReadCsvFile" SCHEMA (Name STRING, Value STRING)
       (
           "$ProjectFolder/data/myconfig.csv" as Filename,
           "TitleRow=false,TimestampColumn=false" as Format
       )
   ) AS CsvFile;
```

This example shows how to read a file identified by the value of a column in an input stream:

```
INSERT INTO MyConfigWindow
SELECT CsvFile.*
FROM
   (SELECT * FROM ControlStream WHERE Command = "RefreshConfig") as
CS ,
   (REMOTE QUERY "ReadCsvFile" SCHEMA (Name STRING, Value STRING)
       (
           CS.Param as Filename,
           "TitleRow=false,TimestampColumn=false" as Format
```

```
    )
) AS CsvFile;
```

Writing Your Own RPC Plugin

Write your own plugin to communicate with remotely callable procedures. This is useful if you want to call a remote procedure that does not use SOAP or HTTP protocols.

A general overview for writing a RPC plugin is below. See the SOAP and HTTP Plugins that provides for more examples. On Microsoft Windows, these are typically located in:

- `C:\Program Files\SybaseC8\Server\sdk\c\examples\c8_rpc_http.cpp`
- `C:\Program Files\SybaseC8\Server\sdk\c\examples\c8_rpc_soap.cpp`

On UNIX-like operating systems, these are typically located in:

- `/home/<userid>/sybasec8/server/sdk/c/examples/c8_rpc_http.cpp`
- `/home/<userid>/sybasec8/server/sdk/c/examples/c8_rpc_soap.cpp`

Every RPC plugin .dll or .so library file must contain at least three functions:

- An initialize function: This function is called exactly once before any calls are made to the execute function. The initialize function may connect to the RPC server, allocate resources (such as memory), or take other appropriate initialization actions. If no initialization is required, the initialize function can return immediately to the caller.
- An execute function: This function is called once for each row processed by the CCL statement. For example, in an application examining email messages for viruses, the execute function might be called once for each email message received by the data stream and passed on to the RPC server.
- A shutdown function: This function is called once when the RPC server shuts down. Typically, a shutdown function closes connections and releases resources. If no shutdown work is required, the shutdown function can return immediately to the caller.

The .dll or .so file that contains these three functions can contain other functions as well. These often include additional functions called by the execute function.

It is recommended, but not required, that you use a separate .dll or .so file for each remote service. A single file may contain multiple sets of initialize, execute and shutdown functions, each of which is used for a separate service. Multiple services that share the same .dll or .so file can each use different initialize, execute and shutdown functions, or can share any number of these functions.

To write your own plugin, use an SDK. The SDK that supports RPC Plugins is *Sybase CEP C/C++ SDK* on page 101.

For more information see *RPC Plugins* on page 264

Compiling an RPC Plugin

The process for compiling a Sybase CEP RPC plugin is similar for both user-created plugins and the plugins provided by Sybase CEP.

For more information, see *RPC Plugins* on page 264.

Remote Database Queries

This section describes remote database queries.

Overview: Querying a Remote Database or Public Window from a CCL Statement

This section provides an overview of the remote database query process.

Remote Database Query Feature

The Sybase CEP remote database query feature enables users to query a remote database or public window with CCL statements.

The term "remote database query" refers either to a database subquery that selects data from a remote database server, or a database statement.

The remote database server refers to a database server that runs outside the Sybase CEP Server process, either on the same computer as Sybase CEP Server, or on a different computer. Sybase CEP's remote database query feature enables CCL statements to execute a query on a remote database server. The query can be a SELECT statement that returns data, or can be some other type of query/statement, such as an INSERT, UPDATE, or DELETE.

A query module can also use a remote database query to query a public window in another module. For more detail, see *Public Windows* on page 65.

The CCL statement can use parameters to pass information from CCL statements to SQL statements. For example, if the current row in the query on Sybase CEP Server has `"DEPT = 'Accounting'"` , then the SQL query that you pass to the remote database server may have a WHERE clause similar to `"... WHERE DEPT = ?DeptName"` . The "?DeptName" tells Sybase CEP Server that the SQL query text should be updated to look like `"... WHERE DEPT = 'Accounting'"` . For more details see Sybase CEP *CCL Reference Guide* .

What Is a Remote Database Query?

The term "remote database query" refers either to a database subquery that selects data from a remote database server, or a database statement.

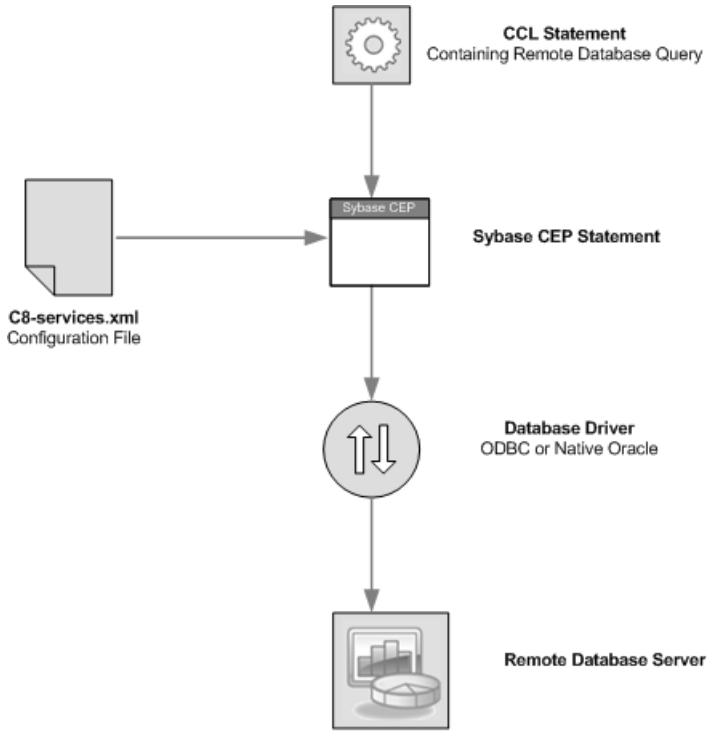
The Components

This section describes the various components of remote database queries.

Remote Database Query Components

An overview of the software components required for Sybase CEP remote database queries and how they interact with one another.

The diagram below shows the components involved in remote database queries and how they are related.



Sybase CEP remote database query functionality requires the following software components:

- A .
- A remote database server.
- A Sybase CEP project (query module) that includes a CCL Statement that queries the remote database server. The CCL statement may be either a Database Statement or a

database subquery. For more information about these CCL statements and clauses, refer to the Sybase CEP *CCL Reference Guide* .

- A section of the `c8-services.xml` file that describes the remote database server as a "service" and allows the Sybase CEP Server to call it. Describe each remote database server in a separate section of the `c8-services.xml` file.
- A database driver (such as an ODBC driver, a Native Oracle driver, or some other appropriate driver) that interfaces between Sybase CEP Server and the remote database server.

These components work together as follows:

1. The CCL Statement invokes the driver by specifying the service name (as identified in the `c8-services.xml` file) and passing the SQL statement.
2. The Sybase CEP Server uses the service name and the information defined for the service in the c8-services.xml file to determine which driver to use.
3. The driver communicates with the remote database server and passes the SQL statement. When appropriate, the driver receives values back from the remote database server and makes them available to Sybase CEP Server, which then makes the values available to the CCL query.

More details about the individual components can be found below.

Remote Database Query CCL Statement

The CCL statement can be either a DATABASE statement or may be a subquery that makes a remote query call.

Below is a pseudocode example:

```
-- Create the table.
EXECUTE STATEMENT DATABASE "LocalOracleDB4"
[[CREATE TABLE Data1 (
Integer_col INTEGER,
Long_col NUMBER(21)
)]]
SELECT SetupStream.command AS command
FROM SetupStream
WHERE command = "CREATE TABLE"
;
```

If there is no filter (such as a WHERE clause), the CREATE TABLE statement is called each time a new row arrives in any of the streams involved in the CCL statement.

Note: As with database subqueries, Sybase CEP Serverhas no knowledge of when information on the remote system is changed. The remote database server cannot notify Sybase CEP Server when data changes; the server does not see updated information until a row arrives in one of the query's streams. The server then executes a statement on the remote database server. There are indirect ways that the remote database server can notify Sybase CEP Server of data changes. For example, the remote database server could use a trigger to call a piece of code that would act as an input adapter to Sybase CEP Server.

For more information about the syntax of these CCL statements and clauses, see the Sybase CEP *CCL Reference Guide* .

The c8-services.xml File

The c8-services.xml file must contain a section for every remote service, before you can use the service in a CCL query.

Each service has a service name (used in the CCL statement) and information about the RPC to be called.

The method of writing a service entry depends on the remote database server you are using. See the *Sybase CEP Installation Guide* for more details.

Sybase CEP Server

Sybase CEP Server executes CCL statements that contain RPCs and invokes the appropriate functions in the plugin to send data to, and receive data from, the remote procedure.

To avoid the overhead of an RPC call, the server can cache information retrieved by the plugin and read data from that cache rather than re-execute the remote procedure call. For information about setting cache expiration times, see the *Sybase CEP Installation Guide* .

The Driver

The driver that you use in remote database queries is based on the database server that you want to access.

You must design and configure the driver to work with the remote database server (for example, Oracle or MySQL,) that you want to use. The driver must be able to contact the database server, pass queries to it, and perform any other necessary actions.

The Remote Database Server

The remote database server runs as an independent process, not as part of Sybase CEP Server. It may run on the same computer as the Sybase CEP Server or on a different computer.

The database server can serve many clients. Furthermore, the client (Sybase CEP Server, plugins, and CCL queries) may be a client of many different database servers. For example, one database server might retrieve historical stock information while another might retrieve current customer information.

Additional Information

A summary of information not contained elsewhere in the chapter.

If the cached information has expired, and a new row arrives in the input stream referenced by the CCL statement containing the remote database query, theSybase CEP Server will call a remote database server. You can set a `c8-services.xml` file parameter to determine how long the server caches responses.

Database subqueries are not blocking calls within Sybase CEP Server. See *Remote Requests, Synchronization, and Performance* on page 64 for more details.

The Sybase CEP *CCL Reference Guide* has information about the syntax of database subqueries and the EXECUTE STATEMENT DATABASE CCL statement.

Additional information about the `c8-services.xml` file is contained in the *Sybase CEP Installation Guide* .

Reading and Writing BLOBs on External Database Servers

A description of situations involving BLOB data.

This section covers the following situations:

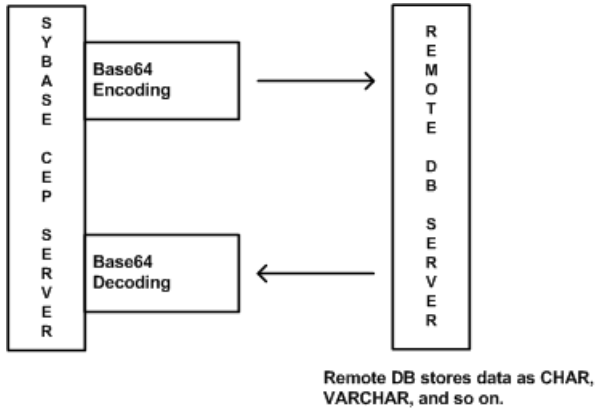
- Reading data from (or writing data to) an external database server that supports the BLOB data type.
- Configuring Sybase CEP Server and wanting to write BLOB data to (or read it from) an external database server that does not support the BLOB data type, but does support a character/varchar/string data type.

BLOB Data Background

Convert BLOB data to a character string in order to store the information on servers that do not typically support the data type.

Many database servers support character data but not BLOB data. You can store BLOB data on these servers by converting it to a character string format before sending it. When reading data back from such servers, you must convert the data from the character string format back to the original Sybase CEP BLOB format.

Sybase CEP Server uses base64 encoding and decoding to convert a BLOB to a character string and vice versa.



In base64 encoding, every 3 bytes of BLOB data is converted to 4 bytes of character string data. This means that the output requires 4/3 as many bytes as the input. If the external

database server character data type has a limit of N bytes per character string, then you can only store 3/4 * N bytes in the server. For example, if the external database server has a limit of 2000 bytes per character string, then the longest BLOB it can store is 1500 bytes, which is expanded to 2000 bytes after base64 encoding. The exact limit may be slightly less than 3/4 * N due to various factors, such as the input not being an exact multiple of 3 bytes.

The DBDriverCharBufferSize limits the maximum size of data that you can write to (or read from) remote database servers.

Connecting and Communicating with External Database Servers that Do Not Support BLOBs

Sybase CEP Server performs base64 encoding and decoding for BLOB data when you store it as character data.

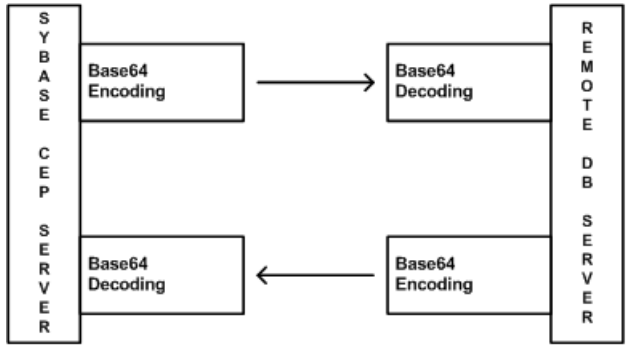
Keep your data within the maximum amount that can be exchanged with the remote database server. The limiting factor may be the amount that the external server character data type can hold, or may be limited by DBDriverCharBufferSize. The longest BLOB you can store on such a server is approximately 3/4 the length of the longest character string that the external server can store or that Sybase CEP Server can exchange with the remote database server.

Connecting and Communicating with Database Servers that Support BLOBs

Connecting and communicating with databse servers that support BLOBs requires either encoding the data in (or decoding it from) base64 format when receiving the data from (or passing it to) external database servers.

Sybase CEP Server sends and receives data in base64 format. To store the data in its original form on the external server, decode the data on the external server before storing it. You can also encode the data on the external server after reading the data from the external server.

The current implementation of the Sybase CEP Server does not provide transparent reading and writing of BLOB information on external database servers. When writing to an external database server, Sybase CEP Server automatically encodes the BLOB data in base64 format even if the external server supports the BLOB data type. Similarly, when reading data from an external database server, Sybase CEP Server automatically decodes data even if the external server supports the BLOB data type. To process the data on the external server, store the data as a proper BLOB on the external server. For example, to process an image on the external server, store the image in its original (BLOB) form so that the image-processing software can interpret it correctly. If you leave the data in base64 format, the image-processing software on the external server can not interpret and process the data.



Remote DB stores data as BLOB

The following is an example of a database subquery that reads data from a BLOB column on an external database server:

```
INSERT INTO FromDB2
SELECT test.id, test.image
FROM
(DATABASE "external1" SCHEMA (id string, image blob)
[[select id, base64_encode(photo_blob) from
testblob T where T.id = '650']])
AS test,
instream;
```

The external database server is told specifically to encode the value in the photo_blob column as a base64 value; this is because Sybase CEP Server expects to receive the data in a base64-encoded form. For this statement to execute properly, the external database server must have a function called base64_encode(). If the database server does not provide this function, you may need to write one yourself.

To write data to a BLOB column on an external database server, tell the external database server to decode the data from base64 into the external server native BLOB format.

```
EXECUTE STATEMENT DATABASE "external1"
[[
insert into testblob2 values(?id,
base64_decode((?image)))
]]
SELECT
  F.id as id,
  F.image as image
FROM
FromDB2 as F
;
```

Encoding and decoding the data consumes resources. If the input or output parameter of the encoding/decoding subroutine has a maximum size limit (typically 2GB), then you are limited to 3/4 of that size. If you only want to read and write the data on the external database server to

store it and do not want to process it as a BLOB on the external database server, store it as character data rather than BLOB data on the external server.

When reading BLOBs from external databases, the maximum size of the BLOB being read in Sybase CEP is defined by the limits of the database driver. When writing BLOBs to external databases, Sybase CEP's DBDriverCharBufferSize setting in the `c8-services.xml` file limits the size of the BLOB, which by default is only 4k. See the *Sybase CEP Installation Guide* for information about this configuration setting.

When reading BLOB data from an external database, transform the BLOB to a base64 encoded string before Sybase CEP acquires the data. Similarly, when writing BLOB data to an external database server, transform the data from base64 format to BLOB format if you want the external database server to store the data in BLOB format.

Different database servers have different packages or functions that you must use to create a base64 encoded string. An example using Oracle's utl_encode.base64_encode() function:

```
INSERT INTO PhotoDB
SELECT InStream2.id, InStream2.image
FROM
InStream1,
(DATABASE "orcl" SCHEMA (id string, image blob)
[[SELECT id,
   -- Tell Oracle to encode the BLOB in base64,
   -- which is the format in which Sybase CEP Server
   -- expects to receive the data.
   utl_raw.cast_to_varchar2(utl_encode.base64_encode(photo))
 FROM testblob T
 WHERE T.id = ?instream1.id]]) AS InStream2;
```

An example of how to write a BLOB into an Oracle database:

```
EXECUTE STATEMENT DATABASE "orcl"
[[
INSERT INTO testblob2 values
 (
 ?id,
 -- Since Sybase CEP sends the blob in base64 format, the external
 -- database server needs to decode the data before storing the
 -- data as a BLOB.
 utl_encode.base64_decode(utl_raw.cast_to_raw(?image))
 )
]]
SELECT
 F.id AS id,
 F.image AS image
FROM
 PhotoDB AS F
;
```

Remote Requests, Synchronization, and Performance

The semantics of the CCL language require that rows be kept in chronological order and processed in chronological order.

A call to a remote server (either an RPC server or a database server) may take a significant amount of time to execute. This depends on such factors as network bandwidth and latency, the load on the remote service, and so on. The delay in getting data back from the remote call may delay the output of rows from the queries.

Sybase CEP Server uses multiple techniques to minimize the delay. Techniques include caching data from the remote service and internal parallelization.

For more information, see the *Sybase CEP Installation Guide* and Sybase CEP Technical Notes.

Caching

Using cached data can reduce server delay caused by processing data.

To reduce the delay, configure the server to cache the data that it retrieves from a remote server and use the cached values rather than re-issue the query. To ensure that the cache is refreshed, set a maximum limit on the length of time that data may stay in the cache. You can also set other caching-related parameters.

See the *Sybase CEP Installation Guide* for more information about caching.

Internal Parallelization

CCL semantics specify that rows be kept in order and processed based on their row timestamps. However, with concurrency enabled, a remote database query can cause the server to issue multiple concurrent database requests on different connections.

These remote queries may not return their results in the same chronological order as the requests were issued. For example, suppose that remote database request "alpha" is issued at 10:00:00, while request "beta" is issued at 10:00:01. Sybase CEP Server may receive the results of the "beta" query before receiving the results of the "alpha" query if the remote server fulfills the "alpha" request in 3 seconds and the "beta" request in 1 second, or if the Sybase CEP Server cache already contains the results of a previous request for beta data. In this case, Sybase CEP Server stores the "beta" remote query results until it receives the "alpha" remote query results and process the rows that needed the data. The server then moves on to process the later rows (those that issued the "beta" request).

Sybase CEP Server ensures that eachinput row is processed in order regardless of whether its requests took a longer or shorter time than other requests issued around the same time. Later input rows have to wait for earlier input rows to be processed. Rows are never processed out of order. The output is the same regardless of when remotely-requested data arrives. When rows

first arrive at the server, they may be re-ordered if the input stream has the MaxDelay or OutOfOrder properties set to allow re-ordering. However, once the data is "inside" the server, the server does not re-order the rows.

Public Windows

Various components can query public windows, including Sybase CEP Projects, SDKs, and the Sybase CEP public windows ODBC driver.

The Sybase CEP Server public window feature enables the rows in a window to be queried by any of the following:

- A separate project running on a Sybase CEP Server that can query the public window using a database subquery or the CCL database statement. See "Database Statement" and "Database Subquery" in the Sybase CEP *CCL Reference Guide* for more information.
- An application written using one of the Sybase CEP SDKs.
- The Sybase CEP-provided ODBC driver for public windows.

For more information about public windows, see "Public Windows" in the Sybase CEP *CCL Reference Guide* .

Sybase CEP Project

If you wish to access the public window from another Sybase CEP project, configure the Sybase CEP Server performing the query so that the server can access the public window as though it were a table in a remote database. This is necessary even if the project with the query and the project with the public window are on the same Sybase CEP Server. Specifically, create a database service entry in the c8-services.xml file; this makes the public windows in the project look like a service that the Database Statement or a database subquery can access. For information about the `c8-services.xml` file and its PublicWindow entries, see the Sybase CEP *Installation Guide* .

The following Sybase CEP SDKs enable you to write a program to read from a public window:

- Sybase CEP C/C++ SDK. For more information, see *Querying a Public Window* on page 262 on page 268.
- Sybase CEP Java 1.5 SDK. For more information, see *Querying a Public Window* on page 290on 298.
- Sybase CEP .NET3 SDK. For more information, see *Querying a Public Window* on page 300on page 311.

Sybase CEP SDK

To query a public window using an SDK, call an SDK function and supply a query written using Sybase CEP SQL syntax. See "Sybase CEP SQL" in the Sybase CEP *CCL Reference Guide* for information about the syntax. The SQL statement must be a SELECT statement, and you must submit only one statement per query. The function returns a result set containing zero or more rows. You can then access each individual row in the result set.

Tip: Test the Sybase CEP SQL query from inside Sybase CEP Studio before using it in an SDK call. This is especially important for SDK calls that use the Query Public Windows in Module command on the Debug menu.

Sybase CEP Public Windows ODBC Driver

Sybase CEP Server includes an ODBC driver for use with public windows. When you set up a new User DSN for the Sybase CEP Public Windows ODBC driver, you provide the ccl URI of the project containing the public window. The URI is in the form ccl://host:port/Project/workspace/project_name, where workspace is the name of the Workspace in which the project runs, and project_name is the name of the project. For example:

```
ccl://mymachine:6789/Project/Default/Filter
```

When you execute a query against the public window, provide the name of the window.

The Sybase CEP Public Windows ODBC driver provides the core level of ODBC compliance, as defined by Microsoft at*http://msdn.microsoft.com/en-us/library/ms714086(VS.85).aspx*.

Note: As you design a system that uses public windows, remember that storing and updating public windows requires Sybase CEP Server to consume both memory and CPU cycles. You may see a reduction in the performance of other queries when you add public windows or add queries of public windows.

Engine Control: Command-line Tools

The chapter describes the actions that can be performed with command-line tools.

Engine Control: Command-line Tools: Overview

A description of various tasks you can perform with command-line tools.

In order to deploy a Sybase CEP application, you must be able to do the following things:

- Start Sybase CEP Server.
- Access a running Sybase CEP Server.
- Create a query module and its associated schema files.
- Compile the query module using the Sybase CEP Compiler.
- Create a workspace on the Sybase CEP Server.
- Execute the query module on the Sybase CEP Server.
- Stop the query module on the Sybase CEP Server.
- Stop the Sybase CEP Server and clean up its resources.

These tasks are executed by one of the following programs:

- The server (c8_server.exe on Microsoft Windows, or c8_server on UNIX-like operating systems).
- The compiler (c8_compiler.exe on Microsoft Windows, or c8_compiler on UNIX-like operating systems).
- The c8_client utility program (c8_client.exe on Microsoft Windows, or c8_client on UNIX-like operating systems). You execute this program with command-line parameters that specify the task to be performed (for example, start executing a query module). The c8_client program translates the command-line parameter(s) into one or more SOAP calls and sends them to the server.

To get the most up-to-date information about the c8_client program, execute

```
c8_client
```

to get a usage message that lists the command-line options.

Note that if you are using the Enterprise Edition of Sybase CEP Engine and you have enabled the User Authentication feature, you will probably need to supply a username and password in order to perform certain operations using c8_client. Use the command line option "--username=<username>" to specify the user name (replace "<username>" with the name of a user who has appropriate permissions). Use the command line option "--password=<password>" to specify the password for that user. If you specify only the "--username" option without the "--password" option, c8_client will prompt you for a password.

Start the Sybase CEP Server

This section describes the command-line parameters and options for starting Sybase CEP Server.

The typical method for starting Sybase CEP Server is documented in the Sybase CEP *Installation Guide* . You can start the Server the same way regardless of whether Sybase CEP applications are deployed from the Sybase CEP Studio or from one of the SOAP interfaces. You can start the Sybase CEP Server from the command line in both Microsoft Windows and UNIX-like environments. The command to invoke the Sybase CEP Server from the command line is:

```
c8_server [options]
```

The following parameters and options can be used with the Sybase CEP Server.

--help	Prints a list of command line parameters.
--version	Prints the current version number.
--service=install	Microsoft Windows Only: This option installs Sybase CEP Server as a Microsoft Windows service.
--service=uninstall	Microsoft Windows Only: This option uninstalls Sybase CEP Server as a Microsoft Windows service.
--service=start	Microsoft Windows Only: This option starts the Sybase CEP Server when it is installed as a service.
--service=stop	Microsoft Windows Only: This option stops the Sybase CEP Server when it is installed as a service and if it is running.
--config=*filename*	Sets the Sybase CEP Server configuration file to the specified file name.
--clean-state	Starts the Sybase CEP Server in a "clean state". All modules that were running or loaded the last time the server was shut down will be cleared. If any modules were using the "persistence" feature, all persisted data will be cleared.

Start the Sybase CEP Server on UNIX-like Operating Systems

Use the command-line to start Sybase CEP Server on a UNIX-like operating system.

On UNIX-like operating systems, the server is started with the command

```
c8-server.rc start
```

The file c8-server.rc is a shell script that sets the environment appropriately and then starts the Sybase CEP Server.

If you try to start the server directly (for example, with the c8_server command) and have not configured the environment properly, you will get an error message such as "error while loading shared libraries". This is an indication that the environment (for example, the LD_LIBRARY_PATH environment variable) is not set properly.

Start the Sybase CEP Server on Microsoft Windows

Use the command-line to start Sybase CEP Server on Microsoft Windows.

On Microsoft Windows, the server is started with a command similar to:

```
C:\Program Files\SybaseC8\Server\bin\c8_server.exe
-c=C:\Program Files\SybaseC8\Server\conf\c8-server.conf
```

This command should all be on one line even though it's not shown on one line in this manual. The "-c=C:\Program Files\SybaseC8\Server\conf\c8-server.conf" indicates which server configuration file the server should use.

If you try to start the server directly with only the command "c8_server.exe", you will likely see a message that includes:

```
license file was not found
```

If you do not specify the configuration file, the Sybase CEP Server will not know where to look for the license file.

Access a Running Sybase CEP Server

Every command or API call that issues one or more SOAP requests to a server needs to know the URI of the Sybase CEP Server.

Some of the API environments hide this detail by taking the URI and creating a connection object so that subsequent requests do not need to provide it every time. Others may require that the URI be specified for every request.

Remember that the form of a server URI is:

```
http://<hostname>:<port>
```

for example,

```
http://omega:6789
```

If your server contains separate managers and containers, use the URI of the manager. Sybase CEP Container Servers are used to run queries, in-process adapters, and user-defined

functions. Sybase CEP Manager Servers "manage" containers by assigning tasks to them and by monitoring their status.

If you are using a secure server (in other words, if you have enabled SSL), use the prefix "https" rather than "http".

Note that a URI for a server or a stream, or any other object accessible with a URI, is useful only when the object is already running.

Create a Project and its Associated Schema Files

This section describes how to create a project (query module) and its associated schema files by using command-line tools.

Sybase CEP Project Files

Describes the CCL and XML components of Sybase CEP project files.

Sybase CEP projects are stored in pairs of files. By convention, these files have extensions of "`.ccp`" and "`.ccl`". These files contain CCL statements and an XML tag structure. Sybase CEP Studio's query editor hides the XML tag structure that wraps the query text with its supporting information (schemas, streams, adapters, parameters and submodule references).

If you choose to build your own query modules, you need to know the XML tag structure. A query module file may be created and edited using the text editor or XML editor of your choice. However, you must create the proper XML tags and content. An XML editor may assist in making sure that all tags are correctly structured. Failure to create a properly structured query module file will almost always result in compile or run-time errors that will need to be corrected. In the vast majority of cases, you should use Sybase CEP Studio or `RegisterQuery()` to create your projects or register your queries.

Note: The XML tag structure used in CEP project files is not fully documented. Furthermore, the internal structure of .ccp and .ccl files is likely to change over the next few releases (starting no later than version 5.1.0). While you may create your own .ccp and .ccl files directly (rather than through Sybase CEP Studio or `RegisterQuery()`) if you wish, we do not recommend this.

Sybase CEP Schema Files

Sybase CEP schema files are stored in files with a conventional file extension of ".ccs". As with query module files, these files use an XML tag structure which is hidden by Sybase CEP Studio's query editor.

Create a Workspace on the Sybase CEP Server

To deploy a Sybase CEP query module, the server must have a workspace that the query module can run inside.

Workspaces are logical divisions within the Sybase CEP Server that are used to isolate groups of query modules.

To create a workspace, execute the command:

```
c8_client --cmd=manager-create-workspace
 --server-uri=<server-uri>
 --workspace-name=<workspace-name>
 --workspace-description=<workspace-description>
```

The c8_client command that performs the Create Workspace operation is called manager-create-workspace. The parameters included with this command are:

Parameter Name	Description
server-uri	The URI of the Sybase CEP Server.
workspace-name	The name of the workspace.
workspace-description	An optional parameter to include a comment string.

The workspace name must conform to Sybase CEP naming rules. If a workspace with the same name has already been created on the Server, then an error will be returned.

Compile a Project or a Schema File

Before you can deploy a project to a Sybase CEP Server, translate the project from its source form (.ccp/.ccl files and .ccs files) to an internal object code form with the Sybase CEP Compiler.

There are two ways to compile a project via the command line:

- By directly invoking the compiler.
- By invoking the compiler through the c8_client program's "register query" command, which compiles the program and starts to execute it.

In each case, the compiled file has the extension .ccx. The next section describes how to compile the project via the CCL compiler. Registering (compiling and executing) the project is discussed in a later section.

Compiling Directly via the CCL Compiler

This section describes the CCL compiler, and how to invoke it on different platforms.

The CCL compiler is a separate program, which can be invoked by the user. The name of the program is c8_compiler (or c8_compiler.exe on Microsoft Windows) and it is stored in the directory `sybasec8/server/bin`. Before invoking the compiler, make sure that your PATH environment variable includes this directory.

On UNIX-like operating systems, you can set the path with a command similar to:

```
PATH=$PATH:/home/<user>/sybasec8/server/bin
```

where "/home/<user>" is replaced by the name of the directory under which the product was installed.

On Microsoft Windows, you can set the path with a command similar to:

```
PATH=%PATH%;C:\Program Files\SybaseC8\Server\bin
```

To compile a CCL program, the compiler must be on the same computer as the CCL source code and the output directory in which you want the resulting .ccx file stored. This does not have to be the same computer as the server is running on.

The general form of the command is

```
c8_compiler InputFile OutputFile
```

for example

```
c8_compiler /home/jsmith/foo.ccp /home/jsmith/foo.ccx
```

Once you have compiled the file, you can start to run it.

The most common case is that you will compile a .ccp file (a project), which can refer to other query modules and to schema files (.ccs files). In some cases, however, you may compile just a single schema file (.ccs). Schema files, as well as project files, must be compiled before they can be used. For example, if you are writing an adapter that needs to read schema information from a file with the SybaseC8SchemaReadFromFile(filename) function, specify the name of the .ccx file that you created by compiling the schema file.

The compiler recognizes several command-line parameters. If you execute the c8_compiler command with the command line parameter -h or --help, or without any parameters, the compiler will return a usage message listing the valid parameters.

The output of the compiler (assuming that there are no errors) is stored in another XML file; by convention, this file has an extension of .ccx. There is only one .ccx file created for each compilation, regardless of how many schema files and submodules are included.

The Sybase CEP CCL Compiler is a stand-alone program that is invoked from the command line:

```
c8_compiler options InputFile OutputFile
```

Where InputFile is the full path and name of a project file, schema file, or plain-text CCL file, and OutputFile is the full path and name of the compiled file you want created.

The following table lists valid parameters for options:

--binding=name=value	Sets the specified module parameter, including stream bindings, to the specified value (more information is provided later in this page). Applies to the module identified with - module, when InputFile is a plain-text CCL file. You can repeat this option as needed to specify the value for multiple module parameters.
--debug	Compiles the debug version of the program.
--guaranteed-delivery=true | false	Enables or disables guaranteed delivery for the project. Defaults to false. Guaranteed delivery guarantees that every row is received by its destination at least once, as long as the software components are running.
--guaranteed-delivery:maximum-age=age	Sets the maximum age, in microseconds, for messages in the guaranteed delivery queue. Messages that exceed this age are removed from the queue without being delivered. Specifying zero for this value sets no limit on the age of messages in the queue (note that this can potentially exhaust available memory). Defaults to 10 minutes. Only valid if guaranteed delivery is enabled.
--guaranteed-delivery:maximum-queue-size=number	Sets the maximum number of messages allowed in the guaranteed delivery queue. Messages that exceed this number are removed from the queue without being delivered. Defaults to zero, which sets no limit on the size of the queue (note that this can potentially exhaust available memory). Only valid if guaranteed delivery is enabled.
--help	Prints a help message.

--import:search-folder=dir	Specifies a directory the compiler should look in for files imported by the project. You can repeat this option as needed to specify additional directories. See "Importing" later in this page for more information.
--instances=N	The number of instances of this program to be spawned on the server.
--max-errror-count=n	Sets the maximum number of errors allowed to n. The compiler will exit if it reaches this limit.
--module=name	Loads the specified module (the main module) from InputFile, which must be a plain-text CCL file. Required when InputFile is a plain-text CCL file.
--name=*progname*	Sets the output program's name to p*rogname*. This name must be unique within a workspace. In most cases, the program name is the same as the name of the source file (minus the .ccp extension). However, if you want to load the same query module more than once within the same workspace, then each instance of the query module must have a unique name.
--no-repository	Use the system root folder as the repository folder.
--optimize:shortcut_and_or=true | false	If set to true, the compiler will use "shortcut" evaluation of expressions that contain AND and OR operators (more information is provided later in this page). If set to false, the compiler will not use shortcut evaluation. The default is true.
--optimize:remove_internal_streams=true | false	If set to true, the compiler removes internal streams (data streams used strictly within a project or query module). Setting this to true means that you won't be able to view the internal streams with Sybase CEP Studio. The default value is false.
--optimize:filter_asap=true | false	If set to true, the compiler will evaluate WHERE and HAVING conditions as early as possible. (This usually maximizes performance.) The default value is true.
--optimize:fold_primitives=true | false	If set to true, the compiler eliminates redundant primitives (more information is provided later in this page). If set to false, elimination of primitives is not done. The default is true. rarely used.

--persistence=true | false	Enables or disables persistence for the project. If set to true, saves to disk state information about the module, including messages. Allows the project to recover after a system failure. Defaults to false. Note that enabling persistence can have a significant negative impact on performance.
--persistence:commit-interval=value	Sets how frequently, in microseconds, Sybase CEP Server should save project state information to disk. Defaults to one second. Only valid if persistence is enabled.
--playback_rate=N	Sets the accelerated playback multiplier to N. Use when input streams are set to use message timestamp and you wish to process the data faster than would otherwise happen based on the message timestamps. Useful for testing or for processing historical data.
--repository=*pathname*	Sets the repository root folder to *pathname*. The compiler uses the repository path name as the root for relative paths, such as to find project and module files. Note that if you omit this parameter, the compiled project references the absolute path to the source files. If you want to connect to the running project using on a computer other than the one on which you compile the project, the absolute path can cause an error when Sybase CEP Studio attempts to locate the source files, unless the directory structure is duplicated on both computers.
--repository-only	Prevents the compiler from accessing files outside the directory specified with the repository parameter. Attempts to import files outside the repository generate an error message. Note that the compiler compares paths for imported files against the path to the specified repository without accessing the file system.
--restart-on-failure=true | false	If set to true, automatically restarts the project (but not any attached adapters) after any fatal error. Defaults to false.
--synchronization=inorder |outoforder | useservertimestamp	Specifies the type of synchronization to use for the project: use the timestamp contained in each message, which must arrive in order (inorder); use the timestamp contained in each message, which can arrive out of order (outoforder); or use the timestamp of Sybase CEP Server when each message arrives (useservertimestamp). Defaults to inorder.

--synchronization:maximum-delay=value	Sets the maximum time, in microseconds, that should wait after each arriving row before sequencing incoming rows from multiple sources. Defaults to one second.
--synchronization:out-of-order-delay=value	Sets the maximum time, in microseconds, that Sybase CEP Server should wait after each arriving row before sequencing incoming rows arriving out of order. Defaults to one second. Only valid if synchronization is set to outoforder.
--version	Prints version information.
--warn:indexes=true | false	If set to true, then the compiler warns when indexes are not used. If the flag is set to false, the compiler does not issue a warning when indexes are not used. The default is false. (Indexes are used to speed up access within large windows.)
--warn:deprecation=true | false	If set to true, then the compiler issues warnings if deprecated parts of the language are used. If set to false, no such warnings are issued. The default is true.
--warn:implicit_conversions=true | false	If set to true, then the compiler issues warnings if implicit data type conversions are used. Ifset to false, no such warnings are issued. The default is true.
--warn:source_without_input=true | false	If true, issues a warning for any named windows or input streams that receive no input from within the window and act as input for other streams or windows. If you specify debug =true, then all named windows act as input to a debug stream for viewing within . Defaults to true.
--warn:ignore-all=true | false	Some operations, such as RegisterQuery, will fail if compilation of the query results in any warnings, even if there are no errors. To force registerQuery to run the query even if there are compiler warnings, set this option to true.
--warn:supress-all=true | false	If true, issue no warnings. Defaults to false.
--workspace=*name*	Sets the output program's workspace name to *name*.

Note: The Sybase CEP CCL compiler can only be invoked as a stand-alone process.

OutputFile

When the compiler compiles a CCL file, it stores the result in a .ccx file. This command-line option allows you to specify the path and name of the file that you want the output written to. When you want to run the program, you will need to specify the location of this .ccx file.

Binding

The binding (-b) option allows you to specify values for both parameters and stream bindings. Consider the following CCL code:

```
create parameter integer Addend;
create input stream StreamIn schema ( Value integer );
create output stream StreamOut schema ( Value integer );
insert into StreamOut select Value + $Addend from StreamIn;
```

The following command sets the value of the parameter Addend to 42, binds StreamIn to the stream named Sink in the project SrcProj, and binds StreamOut to the stream named Source in the project OtherProj:

```
c8_compiler \
   -b=Addend=42 \
   -b=StreamIn=ccl://localhost:6789/Stream/SrcProj/Sink \
   -b=StreamOut=ccl://localhost:6789/Stream/OtherProj/Source \
   file.ccl
```

Importing

If your project imports other files, the compiler searches for those files using two search lists: the command-line search list, composed of all the directories specified with the import:search-folder option, and the project-file search list, as specified in 's compiler options setting and stored in the project file.

The compiler resolves the path to any file that you import with a relative name by searching the following directories, in order, until it locates the file:

1. The directory containing the file that initiated the lookup.
2. Each directory in the command-line search list relative to the compiler's working directory.
3. Each item in the project-file search list relative to the project directory (the directory containing the .ccp file or top-level .ccl file).
4. The project directory itself .
5. Each item in the project-file search list relative to the repository folder (as specified with the repository option).
6. The repository folder itself.

   For example, if a project includes the following statements:

   ```
   import 'x1.ccl';
   import 'x2.ccl';
   import 'x3.ccl';
   import 'x4.ccl';
   import 'x5.ccl';
   import 'x6.ccl';
   ```

Use the following commands to compile the project:

```
cd $ROOT/cwd
c8_compiler -r=$ROOT/repo -I=cmd-lib $ROOT/repo/proj/proj.ccp
```

Assuming that the project-file search list as set in Sybase CEP Studio for this project includes proj-lib, and given the following list of files, the compiler will use the ones highlighted in bold:

```
$ROOT/repo/proj/modules/x1.ccl

$ROOT/cwd/cmd-lib/x1.ccl

$ROOT/cwd/cmd-lib/x2.ccl

$ROOT/repo/proj/proj-lib/x1.ccl
$ROOT/repo/proj/proj-lib/x2.ccl

$ROOT/repo/proj/proj-lib/x3.ccl

$ROOT/repo/proj/x1.ccl
$ROOT/repo/proj/x2.ccl
$ROOT/repo/proj/x3.ccl

$ROOT/repo/proj/x4.ccl

$ROOT/repo/proj-lib/x1.ccl
$ROOT/repo/proj-lib/x2.ccl
$ROOT/repo/proj-lib/x3.ccl
$ROOT/repo/proj-lib/x4.ccl

$ROOT/repo/proj-lib/x5.ccl

$ROOT/repo/x1.ccl
$ROOT/repo/x2.ccl
$ROOT/repo/x3.ccl
$ROOT/repo/x4.ccl
$ROOT/repo/x5.ccl

$ROOT/repo/x6.ccl
```

Primitive Folding

Primitives are instructions that are generated by the compiler and executed by the engine. In some cases, compiling source code (CCL statements) may generate some redundant primitives. Primitive folding merges redundant primitives, decreasing code size and increasing performance.

Shortcutting

Shortcutting AND and OR operators means that the server stops evaluating an expression as soon as the server can determine the result of the expression. For example, if an expression is

"boolCol1 AND boolCol2", and if boolCol1 = false, then it is impossible for "boolCol1 AND boolCol2" to be true, so the server can return FALSE without evaluating the rest of the expression.

In most cases, shortcutting expression evaluation increases performance without changing results. However, there may be cases where you want the entire expression evaluated even if the result can be determined earlier. For example, suppose that you have a WHERE clause that looks like the following:

```
WHERE boolCol1 AND MyBooleanUDF1(col3)
```

If you want `MyUDF1()` to be called for every row (perhaps because the UDF sends information elsewhere or saves information that you'll need later), regardless of the result of the expression, then you should not turn on shortcut evaluation.

In some cases you can re-write expressions to allow shortcut expression evaluation without failing to call every function that should be called. For example, in the simple case shown above, you can simply reverse the order of the operands and then enable shortcut evaluation.

```
WHERE MyBooleanUDF1(col3) AND boolCol1
```

A Note About ADL Files

Before compiling, ensure that the compiler can locate Adapter Definition Language (.adl) files in the Sybase CEP plugins directory to avoid errors.

The compiler reads .adl files which may be provided by Sybase CEP or may be created by customers who write their own input and output adapters. If the .adl files are not in the directory ../plugins (or in any of the other locations that the compiler expects - see *Adapter Definition Language* on page 319 for details), the compiler may give an error message that includes "Cannot locate Plugins folder". To prevent this problem, either run the compiler from the "server/bin" directory or configure your system so that the compiler can find the plugins directory.

Sending the query module to the server

This section describes the command-line procedure and parameters required to send a query module to a server.

Once you compile the .ccx version of the query module, you can send it to the server and start it using the command:

```
c8_client --cmd=start --server-uri=<server-uri>
  --workspace-name=<workspace-name> --program-file=<program-file>
```

The parameters available with this command are:

Parameter	Description
server-uri	The URI of the Server.
workspace-name	The name of the workspace in which the query module should be run.
program-file	The full path and file name of the compiled query module (the .ccx file).

This command loads the query module on the server and starts it at the same time. There are additional SOAP commands that allow you to load and start the query module with separate requests, if desired.

Registering a Project via the c8_client Program

Compile and run a query module by giving the c8_client utility the register-ccl-query command.

The following compilation is done on the server not on the client.

```
c8_client.exe [--config=<config_file>] --cmd=register-ccl-query
--compiler-server-uri=<compiler-server-uri>
--manager-server-uri=<manager-server-uri>
--ccl-query=<path_to_ccl_query_file>
[--compiler-flags=<compiler_flags>]
--workspace-name=<workspace-name> --load=<load_name>
```

The parameters available with this compilation are:

Parameter	Description
config_file	The path to the Sybase CEP Server's configuration file (which is usually named `c8-server.conf` and is usually found under the `server/conf` directory).
compiler-server-uri	The URL of the manager of the server that you want to compile this project. This may be a remote server; it does not have to be a server on the same computer as the c8_client command is running on. The format of this parameter should be similar to: "http://host:port/Compiler", for example, `http:MyComputer:6789/Compiler`.
manager-server-uri	The URL of the manager server that you want to run this project. This may be a remote server; it does not have to be a server on the same computer as the c8_client command is running on. The format of this parameter should be similar to: "http://host:port/manager", for example, " `http:MyComputer:6789/manager` ".

Parameter	Description
path_to_ccl_query_file	The full path and file name of the file that contains the CCL statements. This file should contain only CCL statements and nothing else.
compiler_flags	You can specify the same flags as you would specify on the command line of the compiler. See *Compiling Directly via the CCL Compiler* on page 72.
workspace-name	The name of the workspace in which the query module should be run.
load_name	The name of the project. This name will appear in Studio's Explorer View. This is also the name that you would use if you want to *stop the project* on page 85 from running after you have started it.

Compiling and Running a Project

Compile a query module locally and run it by giving the c8_client utility the compile-and-run command.

This compilation is done on the local client, not on the server:

```
c8_client.exe [--config=<config_file>] --cmd=compile-and-run
--server-uri=<server_uri>
--workspace-name=<workspace_name>
--program-name=<program_name>
--ccl-query=<path_to_ccl_query_file>
[--compiler-flags=<compiler_flags>]
```

The parameters available with this compilation are:

Parameter	Description
config_file	The path to the Sybase CEP Server's configuration file (which is usually named `c8-server.conf` and is usually found under the `server/conf` directory).
server_uri	The URL of the server that you want to run the project on. This may be a remote server; it does not have to be a server on the same computer as the c8_client command is running on. The format of this parameter should be similar to: "http://host:port/manager", for example, `http:MyComputer:6789/manager`.
path_to_ccl_query_file	The full path and file name of the file that contains the CCL statements (.ccl file) or project information (.ccp file).
compiler_flags	You can specify the same flags as you would specify on the command line of the compiler. See *Compiling Directly via the CCL Compiler* on page 72, with the exception of the workspace and program names.
workspace_name	The name of the workspace in which the query module should be run.

Parameter	Description
program_name	The name of the project. This name will appear in Studio's Explorer View. This is also the name that you use if you want to *stop the project* on page 85 after you have started it.

Get Status of an Executing Project

After a project has been started on the Server, it is usually necessary to monitor the status of the project.

There are two ways to monitor the status of a project:

- Poll the Server for status using the appropriate c8_client command.
- Subscribe to the Server's StatusStream for the project.

To poll the Server for status, use the command:

```
c8_client --cmd=status
 --server-uri=<server-uri>
 --workspace-name=<workspace-name>
 --application-name=<application-name>
```

Note: This command should be on a single line. It is split across multiple lines here for readability.

Publishing Data to a Server

This section outlines the use of the c8_client program to publish data to a stream.

To do this, use the "publish" command and specify several command-line parameters. We'll start with an example command and then explain each part of the command. The example is split across multiple lines for clarity, but when you use the command it must all be on one line.

Note that options are prefixed with two dashes, not one.

```
cat data.csv | ./c8_client --cmd=publish
--format=csv
--timestamp-mode=set-current
--format-options='FieldDelimeterChars=^,TimestampColumn=false'
--uri=ccl://hypatia:6901/Stream/Default/SimpleTest/instream
```

In this example,

- The data format is Comma-Separated Values (CSV).

- The Row Timestamp is based on the time that the row arrives, not on a Row Timestamp inside the row.
- The field delimiter character (the character that separates fields within a row) is the caret character "^", rather than the comma ",".
- The row does not contain a Row Timestamp column.
- The c8_client program is ordered to write the data to the stream that has the stream URI `ccl://hypatia:6901/Stream/Default/SimpleTest/instream`.

The following table documents several of the command-line parameters, including the ones used in the example above. For a complete list of parameters that apply to your version of Sybase CEP Server, execute the following command:

```
c8_client --cmd=publish --help
```

Note that the command-line options are case-sensitive.

Parameter	Description
format	Must be one of the three Sybase CEP data stream formats: CSV (Comma-Separated Values), XML, binary. Note that the binary format is not documented, and we recommend that you use the CSV format.
username	The user name. Required if user authentication is enabled.
password	The password of the user you specified. Required only if a username has been specified.
uri	The URI of the stream you want to publish to. For instructions on obtaining the URI of a stream, see *How to Find the URI of a Stream* on page 367.
timestamp-mode	This value may be set to "as-is" or "set-current". If the value is "as-is", the server uses the row timestamp in the data. If the value is "set-current", the server replaces the row timestamp with the current timestamp (as of the time that the row is received by the server).
csv-title-row	If set to true, the server treats the first row as a title row. If set to false, the server treats the first row as a row of data.
csv-timestamp-column	If this is set to true, the first row of data is treated as the Row Timestamp. If set to false, the data is treated as though it has no Row Timestamp.
csv-timestamp-column-format	If this value is set, the server assumes that the format of all timestamp columns, including the Row Timestamp (if present) matches this format. If this value is not set, the server assumes that all timestamp values, including the Row Timestamp (if present) are in the default format (microseconds since midnight January 1, 1970 GMT/UTC). Valid timestamp formats (such as "YYYY-MM-DD HH24:MI:SS.FF") are documented in the Sybase CEP *CCL Reference Guide* .

When you use the "--format-options" command-line parameter, separate the options with commas and put quotation marks around the entire group of format options. On unix-like

operating systems, the quotation marks can be single quotes or double quotes. On MS-Windows, the quotation marks must be double quotes.

For example:

```
cat data.csv | ./c8_client --cmd=publish
...
--format-options=
"FieldDelimeterChars=|,TimestampColumn=false"
```

In this example, we specify that the field delimiter character should be a vertical bar ("|") instead of a comma. We also specify that there is no Row Timestamp column.

The following table shows some of the options that you may use with the "format-options" command-line parameter.

Note that some format-options overlap some of the command-line parameters in the preceding table. For example, you may specify that the input has a title row by using either the format-option `HaveTitleRow=true` or by using the command-line parameter `--csv-title-row=true`. If there is more than one way to specify a particular behavior, use only one of those ways. If you specify conflicting behaviors with different options, the behavior of the product is undefined.

Note that the format options are case-sensitive.

Parameter	Description
TitleRow	Set this to true if the input contains a title row; set it to false otherwise.
HaveTitleRow	Set this to true if the input contains a title row; set it to false otherwise. This option is for backwards compatibility only; otherwise use TitleRow.
TimestampColumn	Set this to true if the first column of input is a Row Timestamp; set it to false otherwise.
HaveTimestampColumn	Set this to true if the first column of input is a Row Timestamp; set it to false otherwise. This option is for backwards compatibility only; otherwise, use TimestampColumn.
TimestampColumnFormat	The format in which timestamp values (including the Row Timestamp, if any) are stored, for example "YYYY-MM-DD HH24:MI:SS.FF". For a complete description of timestamp formats, see the Sybase CEP *CCL Reference Guide* .
NullColumnValue	An explicit string that represents a NULL value. For example, you may use the string "NULL" to represent NULL. An empty field (two field separators with nothing, except possibly whitespace, between them) will also be treated as NULL.
LineDelimiterChars	The character that terminates a line of CSV input (in other words, that terminates a row). The default character is a newline character.

Parameter	Description
LineContinuationChars	The character used to permit a line of CSV data to be continued to the next line. If this character is used, it must be the last character (except whitespace) on the line.
FieldDelimiterChars	The character used to separate columns of CSV data. The default value is a comma.
QuoteChars	The character(s) which may be used to quote CSV strings. By default, either a single quote character (') or a double quote character (") canbe used. Note that the entire string of format options must be enclosed in quotes, so you may need to use the escape character (backslash) to specify a quotation mark within the quoted string. For example, the following specifies that the single quote mark, double quote mark and the back-quote character should all be treated as delimiters for STRING fields: `--format-options="QuoteChars`\"'" ,`
EscapeChars	The character used to indicate that the immediately following character should be interpreted specially.
WhitespaceChars	This allows you to specify a different set of whitespace characters than the default. For example, if you want the tab character (ctrl-I) to be used as a field delimiter, then you do not want it to continue to be treated as a whitespace character. To accomplish this, you might create a set of whitespace characters that exclude the tab character.
TrimWhitespaces	If set to true, whitespace characters outside of quotes are trimmed from STRING fields. If this is set to false, whitespace characters are preserved. Whitespace characters inside quotation marks are always preserved.

Stop Execution of a CCL Project

After a CCL project is successfully started on the Sybase CEP Server, it continues to execute until it is stopped by a client request.

The c8_client command to stop a running project is "stop".

This command should be on a single line. It is split across multiple lines for readability.

```
c8_client [--config=<config file>] --cmd=stop
    --server-uri=<server-uri>
    --workspace-name=<workspace-name>
    --application-name=<application-name>
```

This stops the CCL application <application-name> in Sybase CEP Server at <server-uri>. The parameters of this command are described in the following table:

Parameter	Description
server-uri	The URI of the server.
workspace-name	The name of the workspace.
application-name	The name of the project (query module). This is equivalent to the values specified with the name, program-name or load option when the module was compiled.

Clean Up a Workspace's Resources

When a workspace is no longer in use, elimintate it using the manager-destroy-workspace c8_client command.

```
c8_client [--config=<config file>] --cmd=manager-destroy-workspace
    --server-uri=<server-uri>
    --workspace-name=<workspace-name>
    [--unload-programs-flag=<unload-programs-flag>]
```

This command should be on a single line. It is split across multiple lines here for readability.

This command takes the same parameters as the manager-create-workspace takes.

Parameter	Description
server-uri	The URI of the server.
workspace-name	The name of the workspace.
unload-programs-flag	Indicates whether to unload the program from the workspace.

Stop the Sybase CEP Server

This section describes how to stop Sybase CEP Server.

Stop the Server on UNIX-like Operating Systems

This section describes how to stop the server on a Unix-like operating system using the command-line.

On UNIX-like operating systems, the server is stopped with the command

```
c8-server.rc stop
```

The script c8-server.rc sets the environment appropriately and then starts or stops the server, depending upon the command-line parameter ("start" or "stop").

Stop the Server on Microsoft Windows

This section describes how to stop the server on Microsoft Windows in instances where the server is running as a Microsoft Windows service, and in instances where it is not.

If the server is running as a Microsoft Windows service, you may use the usual Microsoft Windows mechanisms for stopping a service.

If the server is not running as a service, stop the server by pressing ctrl-C in the command window where the server is running. There is no API or other programmatic mechanism for stopping the server on Microsoft Windows.

Implementing Guaranteed Processing

Guaranteed processing allows message delivery in the event of a server crash or lost network connection.

Overview

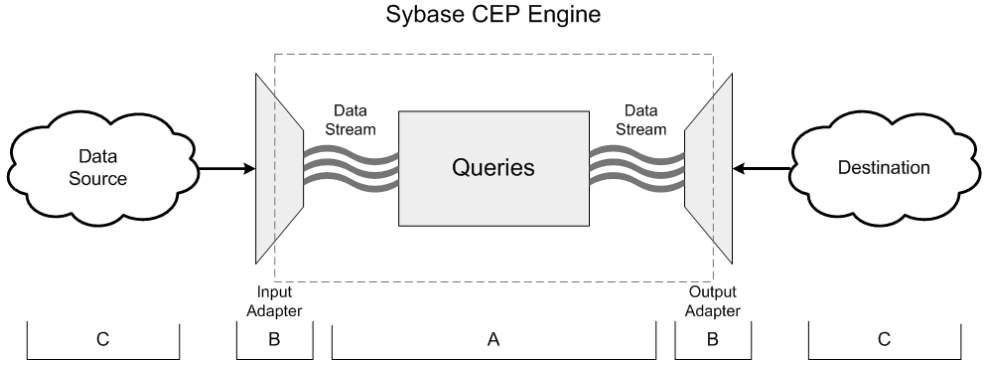
A guaranteed processing sytstem produces exactly the same results (other than timing) with failures as processing would have produced without failures.

Normally when you run a Sybase CEP project or application, Sybase CEP Engine performs best-effort message processing and delivery. In other words, a failure in the system, such as a server crash or lost network connection, may cause messages to be dropped or duplicated. This best-effort processing is fine for many applications, but in some situations, you may need to ensure that every message from your originating data source reaches the final destination exactly once, with no messages dropped or duplicated. A guaranteed processing system ensures that messages are processed exactly once, completely, in order, and with resiliency to failure.

Sybase CEP Engine Application Components

For an application to achieve guaranteed processing, every system component must handle the logic and communication with its neighbor components that ensure guaranteed delivery.

When implementing a guaranteed processing application, you must address every component of the application, not just the Sybase CEP Engine components. The following diagram illustrates the components of an end-to-end application incorporating Sybase CEP Engine:



Messages travel through the application as follows:

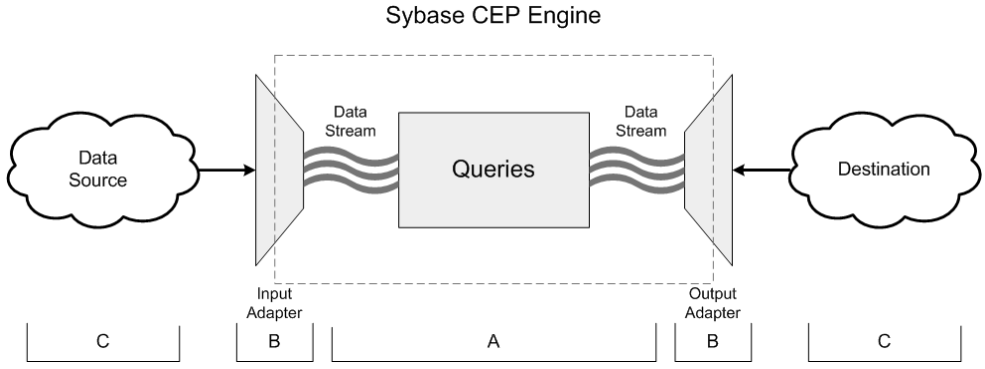
- From the external data source to an input adapter. The adapter may poll the external source or register for notifications or use some other mechanism to receive data.
- The input adapter publishes messages to one or more data streams.
- The stream(s) feed projects and query module(s).
- The query module(s) publish to one or more streams.
- The stream(s) feed an output adapter.
- The output adapter subscribes to the stream(s), processes the messages, converts the data to a format suitable for the final destination, and then transmits the data.
- The destination receives the data and performs whatever actions it is designed to do.

If just one of the components uses some other methodology, then the system as a whole will not perform guaranteed processing. For example, if the data source provides events on a best-effort basis, the entire system can only be categorized as best-effort. Similarly, if the output adapter uses at-least-once processing (messages are never dropped but may be duplicated), then the application as a whole must be considered an at-least-once processor.

Guaranteed Processing Implementation

Guaranteed processing has different requirements for implementation based on the different components of an application. Sybase CEP Engine, adapters, and the source and destination of messages all affect the requirements.

The following diagram organizes the components of an application into three groups, each with different requirements for guaranteed processing:



Sybase CEP Engine

The components labeled with A in the diagram - the streams, projects, and query modules - are part of Sybase CEP Engine and can implement exactly once processing with the help of the Sybase CEP Guaranteed Delivery feature. Guaranteed Delivery is a communications protocol that provides acknowledgements from the recipient to the sender assuring that messages were delivered.

In order to recover after a failure, an application component must be able to re-create its state before the failure and, possibly, retrieve or reproduce some messages. To be able to recover from a Sybase CEP Server failure quickly, consider implementing High Availability.

Note: Guaranteed Delivery can have a dramatic negative impact on performance. Also note that no system is absolutely guaranteed: some types of catastrophic failure can make it impossible (or at least prohibitively expensive) to recover automatically.

See *Sybase CEP Engine Settings* on page 91 for more information about how to configure these components for Guaranteed Delivery.

Adapters

The components labeled with B in the diagram are the input and output adapters. Depending on your application requirements, you may be able to use adapters provided by Sybase CEP or one of Sybase CEP's partners which support guaranteed processing. Enabling guaranteed processing in an adapter implies both the use of the Guaranteed Delivery protocol and the use of algorithms that support guaranteed processing. For example, an adapter might use transactions when you configure it for guaranteed processing, but not use them otherwise. How you enable or configure guaranteed processing is specific to the adapter. If you are using an adapter provided by Sybase CEP, see *Sybase CEP Adapters* on page 377 for more information. If you are writing your own adapter, see *Writing an Adapter for Guaranteed Processing* on page 93 for additional information about implementing Guaranteed Delivery.

Source and Destination

The components labeled with C in the diagram are the data source and destination. Because these are external to Sybase CEP Engine, it's your responsibility to configure them as needed to enable guaranteed processing. You'll need to configure your data source to provide exactly once processing, which is specific to your data source and also dependent on your choice of input adapter. You'll also need to configure the destination to provide exactly once processing. Again, details are specific to your choice of adapter and destination software. See the documentation of the particular software for information about such configuration.

Sybase CEP Engine Settings

Use Sybase CEP Studio or the Sybase CEP Eclipse plugin to enable guaranteed processing on your projects, modules, and streams. You can also specify guaranteed processing settings with dynamic queries if you use a Sybase CEP SDK to build a custom application.

Settings for Projects, Modules, and Streams

Enable and configure Guaranteed Delivery for a project, submodule, or stream in the Properties view of Sybase CEP Studio with setting options.

Use properties of the CREATE MODULE and CREATE STREAM statements to enable Guaranteed Delivery on a module (in the Sybase CEP Eclipse plugin only) or stream. For more

information about the Properties pane, see the "Editing Query Module Properties" section of the Sybase CEP *Studio Guide* . For more information about the CREATE MODULE statement, see the Sybase CEP Eclipse plugin documentation. For more information about the CREATE STREAM statement, see the section of the same name in the Sybase CEP *CCL Reference Guide* .

Sybase CEP query execution is repeatable. Providing the same inputs to multiple executions of the same query will produce the same result. This characteristic allows Sybase CEP Engine to recover from a failure, since it can, in essence, pick up where it left off. Note that there are a few CCL exceptions to repeatability, such as the RANDOM() and NOW() functions, which are designed to return a different result each time you call them.

Persistence

Enable persistence for your application components as part of the guarenteed processing implementation.

The Guaranteed Delivery protocol alone does not ensure exactly once processing: Sybase CEP Engine must save state and message information in order to recover from a failure. In order to provide guaranteed processing, you should enable persistence for your application components. Enable persistence at the project and module level using the same mechanisms described in *Settings for Projects, Modules, and Streams* on page 91. Streams inherit their persistence settings from the parent project or module.

Persistence is enabled on projects and modules whenever guaranteed processing is enabled. However, if the project or submodule does not need to keep non-recoverable state information, then you don’t need to enable persistence. For example, you do not need to enable persistence for a module that only enhances a row with data retrieved from an external database, since the database itself is already persistent.

Start with Clean Slate

Start projects with a clean slate in Sybase CEP Studio.

When you use Sybase CEP Studio to start a project with persistence enabled, you are essentially restarting the project from the saved state. If you want to start fresh, you must click **Start with Clean Slate** in the Sybase CEP Studio **Debug** menu. Making any change to the project also clears the saved state; starting the project after a change is similar to starting with a clean slate.

Writing an Adapter for Guaranteed Processing

To write your own adapter for a guaranteed processing system, incorporate the Sybase CEP Guaranteed Delivery protocol and handle guaranteed processing communication with your data source or destination.

Guaranteed Delivery Mechanisms

A description of how to write an adapter that incorporates the Sybase CEP Guaranteed Delivery protocol.

If you are writing an input adapter, you need to implement the following functionality:

- Establish a connection to Sybase CEP Engine with a unique session ID of your choice.
- Send messages in a batch, each with a batch ID of yoru choice, and wait for acknowledgment of that batch from the subscriber.
- Detect a return after a failure (for example, a lost connection) and re-establish the connection to Sybase CEP Engine with the same session ID. Ask the subscriber for the ID of the last batch it received and begin sending messages again with the next batch.

For an output adapter, you need to implement this functionality:

- Establish a connection to Sybase CEP Engine with a unique session ID of your choice.
- Subscribe for messages and process each batch, tracking the ID of the last batch you processed.
- Detect a return after a failure (a lost connection, for example) and re-establish the connection to Sybase CEP Engine with the same session ID. Notify Sybase CEP Engine of the ID of the last batch of messages you processed before the failure, and then begin processing messages again.

Exactly how you accomplish these tasks is specific to the SDK you use, described in *Sybase CEPC/C++ SDK* on page 101, *Sybase CEPJava 1.5SDK* on page 279, and *Sybase CEP.NET3 SDK* on page 291.

Note: You must also save some information in persistent storage so that you can recover it after a failure. Specifically, an input adapter must be able to retrieve or reproduce the session ID, the ID of the last batch published, and the messages contained in that batch. Depending on your data source, you may be able to reproduce the messages after a failure without having to save them to disk yourself. Similarly, an output adapter must be able to retrieve or reproduce the session ID and the ID of the last batch of messages it processed.

In addition to these Sybase CEP Guaranteed Delivery functions, you must include code in your adapter to handle a variety of situations that are not directly related to the Guaranteed Delivery protocol:

- You must be able to differentiate between a normal start of your adapter and a start after a failure, so that you will know when to retrieve from storage (or re-create) the session ID, batch ID, and (for an input adapter) messages.
- You must be able to handle any special requirements for re-establishing a connection with your data source or destination when you adapter restarts after a failure, including eliminating any duplicate messages.
- You must be able to detect and deal with a lost connection to your data source or destination, and to Sybase CEP Engine, and also detect and deal with the return of that connection.
- You must take care of any special requirements when your data source or destination restarts after a failure.
- You should consider whether your adapter needs to operate in both guaranteed and non-guaranteed modes and, if so, how you will implement both code paths.

Guaranteed Delivery with the Sybase CEP C/C++ SDK

Both Sybase CEP Server and Sybase CEP Studio ship with the Sybase CEP C/C++ SDK. All the C/C++ SDK files are in the directory sdk/c under the Sybase CEP Server or Sybase CEP Studio installation directory.

Note: the discussion in the following pages is limited to functions specific to Guaranteed Delivery. It does not cover general adapter implementation functions. See Sybase CEP *C/C++ SDK* on page 101 for more general information about writing adapters.

Publishing for an In-Process Adapter

An overview of the functionality required to implement guaranteed processing for an in-process adapter and the corresponding Sybase CEP C/C++ in-process API functions you use for each.

Action	Function
Establish a session	C8AdapterConnect
Create and publish batches of messages	C8AdapterSendMessageListAsBatch
Re-establish a session after a failure	Sybase CEP Engine calls the developer-defined reconnect function as specified in the ADL file.
Request the ID of the last batch processed by Sybase CEP Engine	C8AdapterGetLastBatchId

See *In-process Adapter API* on page 161In-process Adapter API and *Signatures of User Functions* on page 176Signatures of User Functions for specific syntax and usage information.

Publishing for an Out-of-Process Adapter

A description of the functionality required to implement guaranteed processing for an out-of-process adapter and the corresponding Sybase CEP C/C++ out-of-process API functions you use for each.

Action	Function
Establish a session	C8PublisherCreateGD
Create and publish batches of messages	C8PublisherSendMessageBatch
Re-establish a session after a failure	C8PublisherCreateGD (only required after an adapter restart)
Request the ID of the last batch processed by Sybase CEP Engine	C8PublisherGetLastBatchId

If you are converting an existing out-of-process adapter to Guaranteed Delivery, you may find it easier to use these functions for publishing messages instead:

Create and publish batches of messages	C8PublisherSendMessage or C8PublisherSendMessages, followed by C8PublisherCommit

See *C/C++ API Interface adapters* on page 141API Interface for specific syntax and usage information.

Sybase CEP provides an example input adapter that implements Guaranteed Delivery, called example_gd_input_adapter.c. You can find it under your Sybase CEP Server or Sybase CEP Studio installation directory, in the sdk/c/examples directory.

Subscribing for an In-Process Adapter

A description of the functionality required to implement guaranteed processing for an in-process adapter and the corresponding Sybase CEP C/C++ in-process API functions you use for each.

Action	Function
Establish a session	C8AdapterConnect
Receive and process batches of messages	C8AdapterGetNextMessagesBatchC8MessageBatchPopMessage
Re-establish a session after a failure	Sybase CEP Engine calls the developer-defined reconnect function as specified in the ADL file.
Specify the ID of the last batch processed	C8AdapterSetLastBatchId

See *In-process Adapter API* on page 161, *Signatures of User Functions* on page 176, and *Message API* on page 121 for specific syntax and usage information.

Subscribing for an Out-of-Process Adapter

A description of the functionality required to implement guaranteed processing for an out-of-process adapter and the corresponding Sybase CEP C/C++ in-process API functions you use for each.

Action	Function
Establish a session	C8SubscriberCreateGD
Receive and process batches of messages	C8SubscriberGetNextBatch
Re-establish a session after a failure	C8SubscriberCreateGD (only required after an adapter restart)
Specify the ID of the last batch processed	Passed as an argument to C8SubscriberCreateGD

If you are converting an existing out-of-process adapter to Guaranteed Delivery, you may find it easier to use these functions for subscribing to messages instead:

Create and publish batches of messages	C8SubscriberGetNextMessage with C8SubscriberGetLastBatchId

If you use these functions, you must keep track of the batch ID so that you can save it to persistent storage at the appropriate time. The function C8SubscriberGetLastBatchId returns the ID of the last complete batch of messages that you have processed with C8SubscriberGetNextMessage. As soon as the ID changes, you should save it to persistent storage so that you can correctly identify the last batch processed after a failure.

See *C/C++ API Interface adapters* on page 141API Interface for specific syntax and usage information.

Sybase CEP provides an example output adapter that implements Guaranteed Delivery, called example_gd_output_adapter.c. You can find it under your Sybase CEP Server or Sybase CEP Studio installation directory, in the sdk/c/examples directory.

Guaranteed Delivery with the Sybase CEP .NET3 SDK

An overview of the locations of Sybase CEP .NET3 SDK files and SDK reference documentation.

Both Sybase CEP Server and Sybase CEP Studio ship with the Sybase .NET3 SDK. All the .NET3 SDK files are in the directory sdk/net3 under the Sybase CEP Server or Sybase CEP Studio installation directory. The SDK reference documentation is in the directory sdk/net5/doc. Open the file `Documentation.chm` to get started.

Sybase CEP ships several examples with the .NET3 SDK, which are under sdk/net3/examples. The examples contain comments to help you understand the purpose of each line of code. For more information about examining, compiling, and running the Sybase CEP .NET3 examples, see *Sybase CEP .NET3 SDK* on page 291.

The file named `Example_13_WorkingWithGuaranteedDelivery.cs` contains an example illustrating how to use the Guaranteed Delivery protocol with the Sybase CEP .NET3 SDK.

Note: the discussion in the following sections is limited to methods specific to Guaranteed Delivery. It does not cover general adapter implementation methods. See *Adapters* on page 13 and *Sybase CEP .NET3 SDK* on page 291 for more general information about writing adapters.

Publishing Guaranteed Delivery in .NET

A description of the functionality required to implement Guarenteed Delivery for an input adapter that uses .NET SDK methods.

Action	Function
Establish a session	CreatePublisherWithGuaranteedDelivery
Create and publish batches of messages	NewBatchOfMessagespublish
Re-establish a session after a failure	CreatePublisherWithGuaranteedDelivery
Request the ID of the last batch processed by Sybase CEP Engine	GetLastBatchId

See the SDK reference documentation in the directory sdk/net3/doc under your installation directory for specific syntax and usage information.

You can see these methods being used in the file `Example_13_WorkingWithGuaranteedDelivery.cs`:

Establish a Session

The following two example lines create a publisher object with guaranteed processing enabled and establishe the connection:

```
publisher =
engineClient.CreatePublisherWithGuaranteedDelivery(inStream,
publisherSessionID);
publisher.Connect();
```

The first parameter of the CreatePublisherWithGuaranteedDelivery method specifies the stream the publisher will write messages to, and the second parameter is the session ID.

Create and Publish a Batch

Here the example creates a batch of messages to be published:

```
IBatchOfMessages batch1 = mf.NewBatchOfMessages(msgList1,
batchID1);
```

The first parameter is the list of messages and the second is the batch ID.

Here the example publishes the batch of messages:

```
publisher.Publish(batch1);
```

Re-Establish a Session

After a (simulated) failure, the example re-establishes the session using exactly the same methods as for the initial connection:

```
publisher =
engineClient.CreatePublisherWithGuaranteedDelivery(inStream,
publisherSessionID);
publisher.Connect();
```

The session ID used here is the same as the ID used the first time, identifying this as a reconnect.

Request Last Batch ID

After reconnecting, the example requests the ID of the last batch of messages Sybase CEP Engine processed:

```
lastBatchID = publisher.LastBatchId;
```

The publisher now knows where to restart processing.

Subscribing with Guaranteed Delivery in .NET

A description of the functionality required to implement Guarenteed Delivery for an output adapter that uses Sybase CEP .NET3 SDK methods.

Action	Function
Establish a session	SubscribeToStreamWithGuaranteedDelivery
Receive and process batches of messages	GetNextBatchOfMessages
Re-establish a session after a failure	ResumeSubscriptionWithGuaranteedDelivery
Specify the ID of the last batch processed	ResumeSubscriptionWithGuaranteedDelivery

See the SDK reference documentation in the directory sdk/net3/doc under your installation directory for specific syntax and usage information.

You can see these methods being used in the file `Example_13_WorkingWithGuaranteedDelivery.cs:`

Establish a Session

The following line of the example not only creates a subscription object with Guaranteed Delivery enabled but establishes the session connection:

```
subscription1 =
engineClient.SubscribeToStreamWithGuaranteedDelivery(
   outStream, subscriberSessionID_1);
```

The first parameter is the stream the subscriber will read messages from, and the second parameter is the unique session ID.

Receive a Batch of Messages

Here the example reads a batch of messages from the data stream:

```
IBatchOfMessages receivedBatch1 =
subscription1.GetNextBatchOfMessages(10000);
```

The parameter is a timeout, in milliseconds.

Re-Establish a Session and Specify Last Batch ID

After a (simulated) failure, the subscriber re-establishes the connection with Sybase CEP Engine:

```
subscription1 =
engineClient.ResumeSubscriptionWithGuaranteedDelivery(
   outStream, subscriberSessionID_1, receivedBatch1.BatchId);
```

The subscriber uses a different method for reconnecting that passes the batch ID of the last batch the subscriber processed, telling Sybase CEP Engine which batch to send next.

Variations of Guaranteed Processing

A guaranteed processing application incorporates both guaranteed processing and persistence. Depending on your application needs, you might want to enable just one of these features in Sybase CEP Engine.

Guaranteed processing without Persistence: If you enable guaranteed processing for the Sybase CEP Engine components of your application without enabling persistence, the application will perform exactly once processing as long as the software components are running. This scenario recovers from failures such as a lost network connection, but not from failures that involve restarting any of the software components. Because the overhead of enabling persistence can have a significant negative impact on performance, you may decide that this option is acceptable for your needs. This variation is called semi-reliable delivery.

Persistence without guaranteed processing: If you enable persistence for the Sybase CEP Engine components of your application without enabling guaranteed processing, the application will still perform best-effort processing, but if Sybase CEP Server restarts, it will

continue processing from the saved state rather than starting fresh. This reduces, but does not eliminate, the possibility of discarded messages.

Sybase CEP C/C++ SDK

This chapter explains how to perform various activities in C/C++, including:

- Creating an out-of-process adapter.
- Creating an in-process adapter.
- Registering a query.
- Compiling a CCL project.
- Creating a UDF.
- Controlling the server - for example, starting or stopping a query module, or getting status information about a server or a query module.
- Embedding a Sybase CEP Server inside another process.

C/C++ Overview

The Sybase CEP SDK in C/C++ programming language includes the following:

- Libraries of functions that you can call.
- Header (.h) files that your source code can #include.
- Source code for some of the adapters that Sybase CEP supplies.
- Sample source code for other purposes, including:
  - An out-of-process input adapter (example_input_adapter.c) and output adapter (example_output_adapter.c).
  - Registering a query (example_register_query.c).
  - A user-defined function (weightedAverage3).
  - A user-defined aggregate function (runningAverage).

You will need to supply a C/C++ compiler.

If you have a .dll file that was generated with a previous version of Microsoft Visual Studio, that .dll file will not work reliably with version 5.1 and later of Sybase CEP Server. You should re-compile the code. This applies to all .dll files that are intended to be linked in with Sybase CEP Server, including:

- In-process adapters.
- UDFs. Note that, throughout this chapter, when we refer to UDFs, we are referring to UDFs written in C/C++, not CCL UDFs, unless explicitly stated otherwise.
- RPC plugins.
- User-defined plugins.

Note: The only C compiler that Sybase CEP supports on Microsoft Windows is the Microsoft Visual Studio .NET 2005 C/C++ compiler with Visual Studio Service Pack 1.

Compiling for 64-bit Microsoft Windows

This section describes how to compile for 64-bit Microsoft Windows, with specific instructions for working with Microsoft Visual Studio.

- Tell the compiler to use the AMD64 code generation tools by running the "vcvarsall.bat" script as shown below (the following should all be on one line, even if it is not displayed as one line in this documentation format):
  "C:\Program Files (x86)\Microsoft Visual Studio 8\VC\vcvarsall.bat" amd64
- If you have a Visual Studio project, make sure the build platform listed in the Configuration Manager is "x64". To do this, follow the instructions below:
  1. Click on "Project" -> Properties.
  2. Click on the "Configuration Manager" button near the top of the property page.
  3. In the drop-down list named "Active solution configuration", select the "Debug" option.
  4. In the "Active solution platform" drop-down list, select "x64".
     a) If you don't see "x64" in the list, then choose "<New...>".
     b) In the "New Solution Platform" dialog box, in the drop-down list named "Type or select the new platform", choose "x64", and in the "Copy settings from" drop-down list, select "Win32".

Note: We have only validated compiling using Visual Studio .NET 2005 (Service Pack 1) on Windows Server 2003 64-bit edition. We have not tried cross compiling from 32 bit platforms

In-process vs. Out-of-process Activities

An overview of in-process and out-of-process activities that are not specific to adapters.

Compiling, registering a query, and server control (for example, stopping execution of a project) are out-of-process activities. All of these out-of-process activities use the Sybase CEP client SDK.

Important: Before you can call any of the client SDK functions, you need to call the client initialization function. (The exact name depends upon which SDK you are using.)

The remaining SDK functionality, such as creating user-defined functions, as well as in-process adapters, uses Sybase CEP Server SDK.

Data Types and Subroutines for UDFs and In-process Adapters

A datatype is one of the internal forms of data used by the Sybase CEP Engine.

The data types are:

C8Int: A signed 32-bit integer.

C8UInt: An unsigned 32-bit integer.

C8Long: A signed 64-bit integer.

C8ULong: An unsigned 64-bit integer.

C8Bool: A true/false switch. Because of C requirements, this has been defined as C8_TRUE and C8_FALSE.

C8Timestamp: Time expressed in microseconds since midnight January 1, 1970 UTC (GMT).

C8Interval: An interval expressed in microseconds.

C8Float: Double-precision floating point numbers (64-bit).

C8CharPtr: A basic character pointer, corresponding to the STRING data type in CCL.

C8Char *: Another form of a basic character pointer, corresponding to the STRING data type in CCL.

C8Blob: A BLOB.

C8BlobPtr: A pointer to a BLOB.

The C SDK treats XML data as a string and thus uses C8CharPtr rather than defining an XML datatype such as "C8XML ".

BLOB data may be stored in one of 3 formats:

- raw: A raw BLOB is stored in its original format, in other words, as a byte-for-byte copy of its value. This is the most compact form and is the form in which Sybase CEP Server internally stores and processes BLOBs.
- hex string: A hex string BLOB is stored as a sequence of hexadecimal digits followed by a byte with the value 0 to terminate the string. This format allows the BLOB to be transmitted and processed as a string of printable characters. This format can also be transmitted or received in situations where you cannot send data that includes control characters. Because each byte of the original data requires 2 bytes of hex digits, and because the string has a single byte as a terminator, the length of the hex BLOB is 2N + 1 bytes, where N is the length of the original BLOB. Although this format is the least compact format, utilities for converting data from raw to hex string format and vice-versa are widely available, and therefore this format is among the most portable formats.
- base64 string: Base64 string encoding is another way of encoding a BLOB as a string. For every 3 bytes of the original BLOB, the base64 string uses 4 bytes. This format may be used when exchanging BLOB information with database servers that expect BLOBs to be sent or retrieved in base64 string format. For more information about base64 string encoding, see *Background* on page 60.

Each Sybase CEP C/C++ SDK function that uses hex string format (either for input parameters or return values) will contain "HexString" somewhere in the name, for example, C8ConvertHexStringToBlob(). If the function uses base64 string format, then the function

will contain "Base64String" somewhere in the name. If the function returns or accepts BLOBs and does not have "HexString" or "Base64String" in the name, then the function handles BLOBs in raw format.

To ensure that the data types are portable, the c8types.h header file includes a typedef for each of these data types.

Error Handling Functions

An introduction to error handling functions, including flexible design options, modifiable error code, and error tracking.

All the error handling functions are available through functions defined in

```
#include "c8messages.h"
```

These error-handling functions are available to both in-process and out-of-process tasks.

The design of error handling permits flexibility in its use. An application may implement error handling to whatever degree necessary. If desired, error handling may be ignored or it may be used after every call. Ignoring error conditions may result in undefined behavior.

You can implement your own error code with associated text messages. This is an alternative way to log errors on the Sybase CEP Engine logger. All calls to C8ErrorSet() result in a log entry in the Sybase CEP system log. All C8ErrorSet() calls are considered equal in the sense that there is no distinction between informational, warning, or fatal messages.

Errors are tracked on a per-thread basis. Once an error condition is detected, the error code will persist until C8ErrorClear() zeroes the error code or until the next Sybase CEP API function is called. Each Sybase CEP API call will reset the error.

A zero error code indicates no errors; this is the initial state.

Refer to c8messages.h for Sybase CEP error codes.

The following commands are available for handling error codes:

- **void C8ErrorClear(); –** Purpose: Clear any existing error code to the zero state. This implies no errors.

  Parameters: None.

  Returns: Nothing.
- **void C8ErrorSet(C8Int error_code, const char *err_text,); –** Purpose: This sets the error code (which may subsequently be read). It also prints both the error code and the message to Sybase CEP Server log.

  The err_text variable can contain not any of the print format specifiers allowed by the NSPR C-language printf() function (for example, "%s", "%d", and so on). See the table below for details. As with the standard C printf() call, the call to C8ErrorSet() should have as many additional parameters as there are print format specifiers. For example:

```
C8ErrorSet(-99, "Err: expected = %d actual = %d", v1, v2);
```

Unlike when calling printf(), you do not need to include a new line to force each message to be put on a separate line in the Sybase CEP log.

The table below shows the print format specifiers used with Sybase CEP data types.

Sybase CEP Data Type	NSPR Print Format Specifier	Meaning
C8Blob	None	There is no format specifier for printing a BLOB. To display a BLOB's contents, you may convert it to a string of hexadecimal digits (using the `C8ConvertBlobToHexString()` function) and then display that string.
C8Boolean	*%d*	Signed integer (note that this will print a number, not a word such as "TRUE" or "FALSE").
C8CharPtr Or C8Char *	*%s*	String.
C8Float	*%f*	64-bit floating point number ("double" on most platforms).
C8Int	*%ld*	Signed 32-bit integer (decimal).
C8UInt	*%u*	Unsigned 32-bit integer (decimal).
C8Interval	*%lld*	This will print the length of an interval in microseconds.
C8Long	*%lld*	Signed 64-bit integer.
C8ULong (UNIX)	*%llu*	Signed 64-bit integer.
C8ULong (Microsoft Windows)	*%I64u*	Unsigned 64-bit integer.
C8Timestamp	*%lld*	This will print the timestamp as a number of microseconds since midnight January 1, 1970, UTC.

For consistency, we recommend that you follow the convention of pairing error numbers and error messages so that you see the same error message each time you see an error number. However, Sybase CEP does not require or check that error codes and error messages are consistently paired. In different calls, the same error code may be used with a different text (and vice versa).

The error you set with C8ErrorSet() will normally remain until you explicitly clear it with the `C8ErrorClear()` function or until it is overwritten by another error.

Note: Errors are "per thread". Each thread sees only its own error codes.

Calls to the Sybase CEP API clear error code when they succeed and set the error code when they fail. We recommend that you check the return codes of Sybase CEP calls to make sure they succeed.

- For calls that return C8Status, the return value of C8_FAIL indicates a failed call, while C8_OK indicates success.
- For functions that return pointers, a return value of NULL indicates that the call failed.
- For void functions, users need to check whether `C8ErrorGetCode()` returns a non-zero value, which indicates an error.

If the Sybase CEP API call failed, the user may get the error code and error text with `C8ErrorGetCode()`, `C8ErrorGetMessageLength()` , and `C8ErrorGetMessageText()`.

Parameters:

- error_code: The error number that you want to set.
- err_text: The text of the error message that you would like to set.

Returns: Nothing.

- **void C8Log(const char *log_text, ...); –** Purpose: This prints the message to Sybase CEP Server log.

  The log_text variable can contain text as well as any of the print format specifiers allowed by the NSPR C-language `printf()` function (for example, "%s" or "%d"). See the table in `C8ErrorSet()` for details. As with the standard C `printf()` call, the call to `C8ErrorSet()` should have as many additional parameters as there are print format specifiers. For example:

  ```
  C8Log("expected = %d actual = %d", v1, v2);
  ```

  Unlike when calling printf(), you do not need to include a new line to force each message to be put on a separate line in the Sybase CEP log.

  Parameters:

  - log_text: The text to write to Sybase CEP Server log.

Returns: Nothing.

- **void C8LogMessage(C8Int i_code, const enum C8LogLevels i_level, const C8Char *i_fmt, ...); –** Purpose: This allows the user to specify message for the log file and control the circumstances under which that message will be printed to the log.

  You may have messages that you want logged only under certain circumstances, such as when debugging a problem. This function allows you to specify a message and the severity of that message. By specifying the appropriate severity for each message, and by setting the LogLevel parameter in the server configuration file, you may control whether a particular message is logged or ignored on a particular run of the server.

  The pre-defined levels are shown in the table below.

C8LogFatalError	-4
C8LogError	-3
C8LogWarning	-2
C8LogStatus	-1
C8LogInfo	0
C8LogDebug	1
C8LogDebug2	2
C8LogTrace	3

  The level of the message, combined with the setting of the LogLevel parameter in the c8-server.conf file, controls whether the message is sent to the log. For example, if c8-server.conf file contains the following:

```
<section name="FileLogger">
  ...
  <preference name="LogLevel">Warning</preference>
  ...
    </section>
```

  then all calls to `C8LogMessage()` that have an i_level of Warning or lower will be printed in the log, and the rest will not. In this example, all FatalErrors, Errors, and Warnings will be printed, while Status, Info, Debug, Debug2, and Trace messages will not be logged.

  The i_fmt variable can contain both text and any of the print format specifiers allowed by the NSPR C-language `printf()` function (for example, "%s" or "%d"). See the table in `C8ErrorSet()` for details about the print format specifiers. As with the standard C `printf()` call, the call to `C8LogMessage()` should have as many additional parameters as there are print format specifiers. For example:

```
C8LogMessage(2001, C8LogDebug, "At start of foo(), param1 =
%d", param1);
```

Unlike when calling `printf()` , you do not need to include a new line to force each message to be put on a separate line in the Sybase CEP log.

Parameters:

- i_code: This number identifies the type of error that occurred.
- i_leve: This identifies the severity of the message, according to the table shown above.
- i_fmt: This contains the text of the error message.

Returns: Nothing.

- **C8Int C8ErrorGetCode(); –** Purpose: returns the most recent error code. The error code and text are maintained on a per-thread basis and persist until cleared or overwritten by another error.

  Parameters: None.

  Returns: The most recent error code.
- **C8SizeType C8ErrorGetMessageLen(); –** Purpose: returns the length of the buffer (in bytes) required to hold the message associated with the error code for the current thread. This length accounts for the space needed for the terminating null. If the return value is zero, there is no text information. It is possible to have a non-zero error code and no associated text if this is what the user has set. Sybase CEP software always associates textual information with an error code.

  Parameters: None.

  Returns: The length of the buffer (in bytes) required to hold the message associated with the error code (for the current thread. Note that this includes the space required for the string termination character.
- **C8Bool C8ErrorGetMessageText(char *o_text, C8SizeType i_buf_size); –** Purpose: the text associated with the current error code is copied into o_text. It is assumed that you have prepared a buffer of adequate size based on the length returned by `C8ErrorGetMessageLen()`. If the buffer is not large enough to hold the message, then the function will copy as much of the message as will fit, taking into account the byte required to terminate the truncated string with a null. The internal copy of the text remains unaffected.

  Parameters:

  - o_text: The buffer into which to copy the error message.
  - i_buf_size: The size of the buffer.

  Returns: C8_TRUE if the entire message was copied, C8_FALSE otherwise.

Memory Management API

An introduction to the memory management functions that can be used in the Sybase CEP Engine.

C/C++ Memory Management API

An introduction to memory management functions that you can use with in-process and out-of-process operations.

- **void* C8Malloc(C8SizeType count); –** Purpose: Allocates an area of memory count bytes in size and returns a pointer to this memory area. Note that the input parameter is an unsigned value (type C8UInt). The memory returned by `C8Malloc()` will contain random data (in other words, the bytes are not initialized to a value such as 0). Except where this documentation explicitly states otherwise, memory allocated by `C8Malloc()` must be de-allocated by `C8Free()` or a memory leak **will** occur. See the warnings in *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

  Parameters:

  - count: The number of bytes to allocate.

  Returns: A pointer to the allocated memory.
- **void* C8Realloc(void *old_mem_ptr, C8SizeType new_size); –** Purpose: Assumes old_mem_ptr has been allocated by `C8Malloc()` and provides for a new memory area of size new_size. The data in the old memory is copied to the new memory. The `C8Realloc()` function follows the same rules as `realloc()` . See the warnings in *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

  Parameters:

  - old_mem_ptr: A pointer to the old memory.
  - new_size: The size of the new block of memory to be reallocated.

  Returns: Nothing.
- **void C8Free(void *mem_ptr); –** Purpose: The memory pointed to by mem_ptr is released. mem_ptr should have been acquired via `C8Malloc()`, `C8Realloc()` , or another SDK function that explicitly states that its returned value may be C8Free'd. See the warnings in *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

  Parameters:

  - mem_ptr: A pointer to the memory to be freed.

  Returns: Nothing.

Although the memory management API is the same for in-process and out-of-process operations, some special cautions apply when these are used with in-process operations. These cautions are explained in the following section.

Notes on Allocating and Deallocating Memory in In-process Code

Unless a function's description states otherwise, if you allocate a piece of memory, then you are responsible for de-allocating that memory, and if the server allocates a piece of memory and returns it to you, the server is responsible for de-allocating the memory.

For each piece of memory that is allocated by either you or the server, the de-allocation method must correspond to the allocation method that you (or the server) used. For example, if you allocate a piece of memory using `C8Malloc()`, then the memory should be freed using `C8Free()` rather than `free()` or some other de-allocation method. If you use the wrong de-allocation method for a particular piece of memory, the results are undefined.

Similarly, if you allocate a piece of memory that the server is responsible for de-allocating, the memory must be allocated using the appropriate function so that the server can de-allocate it correctly.

These rules apply to all code in IPAs and UDFs (including user-defined aggregate functions).

The following table shows when to use each type of de-allocation method:

Allocated By	Deallocated By	Language
`C8Malloc()` or `C8Realloc()`	`C8Free()`	C or C++
`malloc()` or `realloc()`	`free()`	C or C++
`new (C++ memory allocation)`	`delete (C++ memory deallocation)`	C++
`C8MessageCreate()` or `C8MessageCreateWithSize()C8MessageCreateWithSize()` or `C8AdapterReceiveMessage()` or `C8AdapterReceiveMessageWait()`	`C8MessageDestroy()`	C or C++

Warning! You must use the appropriate de-allocation function for each piece of memory. For example, use `C8Free()`, not `free()`, to free a piece of memory that you allocated with C8Malloc().

In general, if a function returns a pointer to object A that is part of another object B, then you should not specifically de-allocate object A; A will be de-allocated when B is de-allocated. In

the Sybase CEP C/C++ SDK function prototypes, objects that are part of other objects are usually marked as "const" ; for example, in the function prototype below, the return value is "const C8Schema *", which indicates that the returned schema is part of a C8Publisher object, and therefore should not be de-allocated separately.

```
const C8Schema * C8PublisherGetSchema(...);
```

C/C++ Data Conversion Functions

The Data Conversion Functions permit customers to change data from one data type to another - for example, to convert the string "2006-02-01 09:00:00.000000" into a value that is formatted as a TIMESTAMP.

These functions may be used in both in-process and out-of-process code.

Conversion API

The API for all conversions requires inclusion of the `c8conversions.h` header file.

Include the `c8conversions.h` header as follows:

```
#include "c8conversions.h"
```

The actual code to convert the data types is in a shared object library (".so") file on most UNIX-like operating systems or a Dynamic Link Library (".dll") file on Microsoft Windows systems. When you compile and link your code, include the appropriate library as part of the link command.

ToString() datatype

All of the ToString() functions take an input data value and convert it to a string. The string is stored in a user-supplied data buffer with a specified max size. All data types except C8Bool require a conversion format.

To use the Sybase CEP default format, pass NULL as the format argument. You are responsible for supplying the buffer to save the output.

If conversion is successful, the function returns `C8_TRUE`. If conversion is unsuccessful, then the function returns `C8_FALSE`; the content of the output buffer is undefined; and the Sybase CEP Engine log will contain an error message. If the output buffer is too short, the output string is silently truncated and `C8_FALSE` is returned.

- **C8Bool C8ConvertIntToString(C8Int i_value, const C8Char *format, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts an integer to a string. The format defaults to "%d", the standard C-language format specifier for integer. You may provide any C-style integer format.

  Parameters:

  - i_value: The value to convert to a string.

- format: A format specifier indicating the desired format of the resulting string value.
- o_buf: The location into which to store the string. The user must have allocated enough memory to hold the desired string beforehand.
- o_buf_size: The length of the buffer.

Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.

- **C8Bool C8ConvertLongToString(C8Long i_value, const C8Char *format, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts a long to a string. The format defaults to "%lld". Users may provide any C-style long format.

  Parameters:

  - i_value: The value to convert.
  - format: The format specifier indicating the desired format of the resulting string value.
  - o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.
  - o_buf_size: The length of the buffer.

  Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.
- **C8Bool C8ConvertBoolToString(C8Bool i_value, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts a boolean to a string. The result will be the string "true" or "false". If i_value is non-zero, the result will be "true". If i_value is zero, then the output will be "false".

  Parameters:

  - i_value: The value to convert.
  - o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.
  - o_buf_size: The length of the buffer.

  Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.
- **C8Bool C8ConvertTimestampToString(C8Timestamp i_value, const C8Char *i_format, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts a timestamp to a string. If the i_format parameter is invalid, the conversion may fail.

  Parameters:

  - i_value: The value to convert.
  - i_format : A format specifier indicating the desired format of the resulting string value, for example, "YYYY-MM-DD HH24:MI:SS.FF". If the format parameter is a null pointer, the output is in microseconds. To specify another format, please refer to the Sybase CEP *CCL Reference Guide* 's description of valid formats for TIMESTAMP data type. If the time zone is not specified, the format will default to using local time.
  - o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.

- o_buf_size: The length of the buffer.

Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.

- **C8Bool C8ConvertTimestampTimezoneToString(C8Timestamp i_value, const C8Char *i_format, const C8Char *i_timezone, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts an internal timestamp to a string while applying appropriate timezone offsets. Conversion may fail if the format or timezone is invalid. Please refer to the time formatting specifications for supported formats. The timezone abbreviations are defined in the `c8_timezones.csv` file in the plugins directory. See the discussion of time zones and the `c8_timezones.csv` file in the Sybase CEP *CCL Reference Guide* for more details.

  Parameters:

  - i_value: The value to convert.
  - i_format: A format specifier indicating the desired format of the resulting string value, for example, "YYYY-MM-DD HH24:MI:SS.FF". If the format parameter is a null pointer, then the output is in microseconds.
  - i_timezone: The timezone abbreviation, for example, "PST" for Pacific Standard Time.
  - o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.
  - o_buf_siz: The length of the buffer.

  Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.
- **C8Bool C8ConvertIntervalToString(C8Interval i_value, const C8Char* i_format, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts an interval to a string. The parameter i_format is usually NULL, in which case the output interval in o_buf will be in the standard Sybase CEP interval form, similar to +1 23:45:67.123456 (days, hours, minutes, seconds, and fractions of a second). For alternative formats, see the `NSPR prprf.h` file, which lists the formats accepted by the underlying PR_snprintf code.

  Parameters:

  - i_value: The value to convert.
  - i_format: A format specifier indicating the desired format of the resulting string value.
  - o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.
  - o_buf_size: The length of the buffer.

  Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.
- **C8Bool C8ConvertFloatToString(C8Float i_value, const C8Char *i_format, C8Char *o_buf, C8SizeType o_buf_size); –** Purpose: This function converts a float to a string. The format uses the default of "%lg". Users may provide any C style format that is valid for the C "double" data type (64-bit floating point value), which corresponds to the CCL FLOAT type.

Parameters:

- i_value: The value to convert.
- i_format: A format specifier indicating the desired format of the resulting string value.
- o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.
- o_buf_siz: The length of the buffer.

Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.

- **C8Bool C8ConvertBlobToHexString(const C8Blob * i_val, C8UInt i_blob_size, C8Char * o_buf, C8SizeType i_buf_size); –** Purpose: This function converts a raw BLOB to a string of hexadecimal digits. The caller must provide a buffer for the string large enough to contain the converted string (including the terminating null). Since each byte of the blob will take two characters (hex digits), the size of the buffer must be 2 * i_blob_size + 1.

  Note that since strings are limited to 2^32 bytes on many platforms, the longest Blob you can convert to a hex string on those platforms is 2^31 - 1 bytes. Blobs that are 2GB or longer cannot be converted.

  C8SizeType is typedef'd in c8types.h.

  Parameters:

  - i_value: The value to convert.
  - i_blob_size: The number of bytes in the raw BLOB.
  - o_buf: The location in which to store the string. Allocate the required memory to hold the desired string beforehand.
  - i_buf_size: The length of the buffer.

  Returns: `C8_TRUE` on success, `C8_FALSE` otherwise.

StringToDatatype()

All the StringToDatatype functions take an input string and output a parsed value of the appropriate data type.

If the parse cannot be performed, the functions return C8_FALSE. An invalid parse results in an undefined result. A valid parse returns C8_TRUE and the parsed value of the appropriate type in o_out.

Data can be preceded or followed by whitespace. The entire string is assumed to contain the data. The parse is not as in sscanf() where "123XX456" produces the integer 123. In Sybase CEP, parsing "123XX456" as a number will result in a parse error. A string of " 123\t " (where \t is the tab char), is perfectly acceptable since whitespace may lead and follow a numeric value.

The following functions can be used to convert strings to datatypes:

```
C8Bool C8ConvertStringToInt      (const C8Char *s, C8Int *o_out);
```

```
C8Bool C8ConvertStringToLong     (const C8Char *s, C8Long *o_out);
C8Bool C8ConvertStringToBool     (const C8Char *s, C8Bool *o_out);
C8Bool C8ConvertStringToTimestamp(const C8Char *s,
C8Char*i_format,                                   C8Timestamp *o_out);
C8Bool C8ConvertStringToFloat    (const C8Char *s, C8Float
*o_out); // Convert a "raw" blob to a "hex string" blob.C8Bool
C8ConvertHexStringToBlob  (const C8Char *s, C8Blob
*o_out,                                    C8SizeType
*o_size,                                    C8SizeType i_buf_size);
```

Note that the `C8ConvertStringToTimestamp()` function is the only one of these StringToDatatype() functions that allows you to specify a format. If the format parameter is a null pointer, the output is in microseconds. To specify another format, for example, "YYYY-MM-DD HH24:MI:SS.FF", please refer to the description of valid formats for TIMESTAMP data type in the Sybase CEP *CCL Reference Guide* . If the format is invalid, conversion may fail.

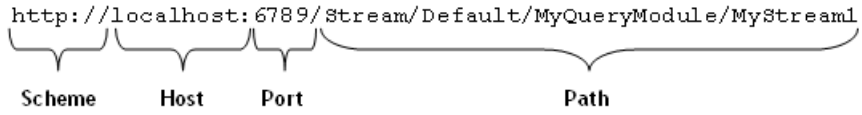
Note that since 2 hex digits are converted to a single byte in the BLOB, and since the trailing null (string terminator) is not needed as part of the blob, the o_out parameter to the `C8ConvertHexStringToBlob()` function only needs to be CEIL((N-1)/2) bytes long, where N is the length of the hex string.

Note: C8SizeType is typdef'd in c8types.h.

Miscellaneous C/C++ functions

This section describes miscellaneous C/C++ functions, their applications, and parameters.

- **C8Bool C8ParseURL(const C8Char *url, C8Char *scheme, C8SizeType scheme_size, C8Char *host, C8SizeType host_size, C8Int *port, C8Char* path, C8SizeType path_size); –** Purpose: Given a Uniform Resource Locator (URL) of the form:

  parse the URL into the various components:

  - scheme: The protocol, for example, "ccl://" or "http://"
  - host: The host computer
  - port: The port number
  - path: The remainder of the URL after the port number.

  If the parse is successful, C8_TRUE is returned; otherwise, C8_FALSE is returned. You must provide C8Char arrays of adequate size to contain the output of the parse. Results will be truncated if the arrays are too small. If the return value is C8_FALSE, the results are undefined. Suggested sizings are shown in the code fragment below:

  ```
  C8Char scheme[8];
  ```

```
    C8Char host[256];
    C8Int port = 0;
    C8Char path[256];
    if ( ! C8ParseURL(url, scheme, sizeof(scheme), host,
sizeof(host),
      &port, path, sizeof(path)))
  { ... }
```

Parameters:

- url: The URL that you would like parsed.
- scheme: The scheme specifies the protocol, for example, "ccl://" or "http://".
- scheme_size: The number of bytes in the scheme variable.
- host : The name of the host computer specified in the URL.
- host_size: The number of bytes available to store the host name.
- port: The port number specified in the URL.
- path: The remainder of the URL after the port number.
- path_size: The number of bytes available to store the path.

Returns: C8_TRUE if the URL is parsed successfully; otherwise, C8_FALSE.

Generic Functions Available in Out-of-process and In-process Tasks

This section describes generic functions for out-of-process and in-process tasks, their purposes, and their parameters.

- **C8Timestamp C8Now(); –** Purpose: This returns the current time. The returned value is a standard TIMESTAMP value; in other words, it is the number of microseconds since midnight January 1, 1970 UTC/GMT. Although the time is returned as a number of microseconds, the actual precision depends upon the platform. On Microsoft Windows, the typical return of `C8Now()` is 15 milliseconds (15,000 microseconds).

  Parameters: None.

  Returns: The current time as a C8Timestamp.
- **C8Timestamp C8HighResolutionNow(); –** Purpose: This returns the current time. The returned value is a standard TIMESTAMP value; in other words, it is the number of microseconds since midnight January 1, 1970 UTC/GMT. This function uses high-resolution timers for microsecond accuracy on the platforms that support such timers.

  Parameters: None.

  Returns: The current time as a C8Timestamp.
- **void C8Sleep(C8Interval us_delay); –** Purpose: The thread sleeps for the specified amount of time. The header file `c8types.h` contains definitions that may be useful in specifying durations; for example, to delay 3 seconds, use 3*C8PerSecond.

  Note that small sleep intervals are probably inappropriate depending on your CPU speed, system clock resolution, and system loading. In general, resolutions less than 100

milliseconds are very imprecise. Even for larger values, the actual length of the sleep will likely not be exactly what was specified.

Parameters:

- us_delay: The number of microseconds to delay.

Returns: Nothing.

APIs Used for Out-of-process Adapters and Control Programs

The following APIs are used both by out-of-process adapters (adapters that run as separate programs rather than as part of the server process) and programs that do control activities such as compiling and executing CCL code.

c8client.h

An introduction to c8client.h functions, including their purposes and parameters.

The c8client.h library contains the following functions:

- **C8Status C8ClientSDKInitialize(const C8Char * i_progname, const C8Char * i_pref_file); –** Purpose: This function initializes the SDK. This function must be called before calling any of the functions in the out-of-process adapter API or the control API. Call the function C8ClientSDKShutdown() when the SDK is no longer needed.

  Parameters:

  - i_progname: Program name used for logging.
  - i_pref_file: Path to preferences file to use. Pass NULL to specify no preferences file and to use defaults.

  Returns: C8_OK if successful, C8_FAIL otherwise.

  Warning! If you are writing an out-of-process adapter that uses SSL, you must create a subdirectory named **secure** in the directory containing your executable. To store your security certificate information, copy the **secure** directory into the same directory as your executable from the top-level directory of either the Sybase CEP Server or Sybase CEP Studio installation. If the Sybase CEP application running when you attempt this step, you will see an error similar to the following:

  CRITICAL ERROR: Error: Error while generating certificate (Reason='Error: failed to initialize internal key slot')

- **C8Status C8ClientSDKShutdown(); –** Purpose: This function cleans up after using the out-of-process adapter SDK or the server control SDK. No SDK functions may be called after the call to C8ClientSDKShutdown().

  Parameters: None.

Returns: `C8_OK` on success, `C8_FAIL` otherwise.

APIs Used for In-process and Out-of-process Adapters

The following APIs can be used with both in-process and out-of-process adapters.

Schema API

An introduction to the schema-related APIs in the header file c8messages.h, including their purposes and parameters.

- **C8UInt C8SchemaGetColumns(const C8Schema *i_s); –** Purpose: For a given schema, returns the number of columns in the schema.

  Parameters:

  - i_s: A pointer to a schema object.

  Returns: The number of columns in the schema.
- **C8_TYPES C8SchemaGetColumnType(const C8Schema *ptr, C8UInt col_ndx); –** Purpose: For a given schema and a particular column index in that schema, returns the type of information contained in that column. The type of information can be used in a switch statement or other control code. Column indexes are 0-based, not 1-based - in other words, they range from 0 to C8SchemaGetColumnsCount()-1.

  Parameters:

  - ptr: A pointer to a schema object.
  - col_ndx: Indicates which column to get the info for. Note that column indexes are 0-based, not 1-based.

  Returns: The type of information contained in that column. This is one of the enumerated data types defined for C8_TYPES; see c8types.h for details. The function returns C8_INVALID if col_ndx is out of range or if another error occurs.
- **C8_TYPES C8SchemaGetColumnTypeByName(const C8Schema *ptr, const char *i_fieldname); –** Purpose: For a given schema and a particular column name in that schema, return the type of information contained in that column. The definition of C8_TYPES is in c8types.h. The type information can be used in a switch statement or other control code. Column indexes are 0-based, not 1-based, and they range from 0 to `C8SchemaGetColumnsCount()-1`.

  Parameters:

  - ptr: A pointer to a schema object.
  - i_fieldname: The name of the field for which you want the data type.

Returns: The type of information contained in that column. This is one of the enumerated data types defined for C8_TYPES; see c8types.h for details. The function returns C8_INVALID if col_ndx is out of range or if another error occurs.

- **const C8Char* C8SchemaGetColumnName(const C8Schema *ptr, C8UInt col_num); –** Purpose: For a given schema and a particular column index in that schema, return the name of that column.

  Parameters:

  - ptr: A pointer to a schema object.
  - col_num: The index of the column you want the information for.

  Returns: The name of the column. The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer. The returned pointer is valid as long as the schema pointer is valid.
- **C8Status C8SchemaGetColumnPosition(const C8Schema *ptr, const C8Char *col_name, C8UInt *o_ndx); –** Purpose: Given a schema pointer and a column name, return the corresponding column number. Column names are 0-based, not 1-based.

  Parameters:

  - ptr: A pointer to a schema object.
  - col_name: The name of the column for which you want the info.
  - o_ndx: A pointer to a C8UInt that can hold the column number.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Schema *C8SchemaCreate(void); –** Purpose: This creates a schema that is empty (i.e. for which no columns are defined). You can add columns by using the `C8SchemaAddColumn()` function. The schema must be destroyed with `C8SchemaDestroy()`.

  Parameters: none.

  Returns: A new schema pointer, or NULL if there was an error.
- **C8Status C8SchemaAddColumn(C8Schema *i_schema, C8_TYPES i_type, const C8Char *i_name) –** Purpose: Adds a column to the schema. The new column will have the specified name and data type.

  Parameters:

  - i_schema: The schema you want to add a column to.
  - i_type: The data type of the new column.
  - i_name: The name of the new column.

  Returns: C8_OK if success, C8_FAIL otherwise.
- **C8Schema *C8SchemaReadFromFile(const C8Char *i_filename) –** Purpose: Reads a schema from a file. The input parameter is the name of the file to read the schema from. This file must contain a compiled schema (a "tuple descriptor"). For information about

how to compile a .ccs schema file and generate a .ccx file, see *Compile a Project or a Schema File* on page 71.

Parameters:

- i_filename: The name of the file from which to read the schema.

Returns: Returns a pointer to the schema if successful, NULL otherwise. When you are done using the schema, the returned pointer must be freed with a call to C8SchemaDestroy().

- **C8Schema *C8SchemaReadFromString(const C8Char *i_str) –** Purpose: Reads a schema from a string. The input parameter is the string to read the schema from. The stream must contain the XML text representation of a compiled schema (in other words, a "tuple descriptor").

  Parameters:

  - i_str: The string from which to read the schema.

  Returns: A pointer to the schema if successful, NULL otherwise. When you are done using the schema, the returned pointer must be freed with a call to `C8SchemaDestroy()`.
- **void C8SchemaDestroy(C8Schema* i_schema) –** Purpose: Destroys a schema by de-allocating its memory. Any constant pointers returned by calling functions with this schema are invalidated upon return from this method.

  Parameters:

  - i_schema: A pointer to the schema to destroy.

  Returns: Nothing.
- **C8Status C8SchemaToXML(const C8Schema *i_s, C8Char **o_buf) –** Purpose: Converts a schema to a string containing the schema's XML representation. The representation is of a compiled schema (a tuple descriptor). This function allocates enough memory for the string, and the pointer returned through the o_buf parameter needs to be freed with a call to `C8Free()`.

  Parameters:

  - i_s: A pointer to a schema.
  - o_buf: A pointer that can be set to point to a string containing the XML representation of the schema.

  Returns: C8_OK if success, C8_FAIL otherwise (in which case o_buf is unchanged).
- **C8Status C8SchemaWriteToFile(const C8Schema *i_s, const C8Char *i_fn) –** Purpose: Writes a schema to a file in tuple descriptor representation.

  Parameters:

  - i_s: A pointer to the schema.

- i_fn: The name of the file to write the schema to.

Returns: C8_OK if success, C8_FAIL otherwise.

- **C8Bool C8SchemaCompare(const C8Schema *i_s1, const C8Schema *i_s2) –** Purpose: Return a value indicating whether two schemas are identical.

  Parameters:

  - i_s1: First schema.
  - i _ s2: Second schema.

  Returns: C8_TRUE if the schemas are identical, C8_FALSE otherwise.

Message API

An introduction to the message-related APIs in the header file c8messages.h, including their purposes and parameters.

A C8_MESSAGE_TYPE describes whether a message is entering or exiting an internal window.

Most adapters will use C8_MESSAGE_POSITIVE for all messages. A normal message is a positive message. In other words, it corresponds to a row of data in the stream.

The other types of messages are not currently intended for use with adapters.

In general, when a "const *" is returned for data associated with an object (for example, a schema associated with a message), the validity of the returned pointer is limited to the lifetime of the pointer's owner (for example, the message). Attempts to de-reference the pointer after the owner no longer exists may result in a runtime exception.

- **C8Message* C8MessageCreate(enum C8_MESSAGE_TYPES msg_type, const C8Schema *schema_ptr) –** Purpose: Creates an empty message of the specified message type with the specified schema. This is the initial call in composing a message.

  Parameters:

  - msg_type: In almost all cases, this is C8_MESSAGE_POSITIVE. For a list of other message types, see the definition of C8_MESSAGE_TYPES in the file `c8messages.h`.
  - schema_ptr : A pointer to the schema that you want to define the structure of the message.

  Returns: A pointer to the newly created message. When you are done using this message, destroy it by passing it to `C8MessageDestroy()` .

  See also the function `C8MessageCreateWithSize()` below.

- **C8Message* C8MessageCreateWithSize(enum C8_MESSAGE_TYPES msg_type, const C8Schema *schema_ptr, C8UInt fields_size) –** Purpose: Creates an empty message of the specified message type with the specified schema and with the amount of

memory allocated. This is the initial call in composing a message. By specifying the size of the entire message, performance may be increased.

Parameters:

- msg_type: In almost all cases, this is C8_MESSAGE_POSITIVE. For a list of other message types, see the definition of C8_MESSAGE_TYPES in the file `c8messages.h`.
- schema_ptr: A pointer to the schema you want to define the structure of the message.
- field_size: A user estimated size of the entire message.

To calculate the fields_size, sum up the sizes of all of the fields, where the size of each field is the number of bytes of data stored in the field rounded to the closest larger multiple of 4, plus 4 bytes. For example, if you have a value that requires 5 bytes, then the total size for that field is 8 + 4 (5 rounded up to 8, plus 4) - a total of 12 bytes.

Estimates that are greater than the actual resulting messages size will result in unused memory. Estimates that are too low will result in automatic adjustments upward to the correct message size. Adjustments upward in size may decrease performance.

Returns: A pointer to the newly created message. When you are done using this message, you must destroy it by passing it to `C8MessageDestroy()`.

See also the function `C8MessageCreate()`.

void C8MessageDestroy(C8Message *msg_ptr)

Purpose: Destroys the message pointed to by msg_ptr. All messages created by the user or received by the `C8AdapterReceiveMessage()` or `C8AdapterReceiveMessageWait()` call must be destroyed after it has been dispatched to the Sybase CEP Serveror else a memory leak will occur.

Parameters:

- msg_ptr: A pointer to the message to be destroyed (de-callocated).

Returns: Nothing.

- **enum C8_MESSAGE_TYPES C8MessageGetType(const C8Message *msg_ptr) –** Purpose: The message type is returned. There are three different categories of messages. There are positive messages, which are simply normal messages. There are also negative messages, which represent rows removed from window. The third category of message is control messages. Some of these are only used internally, and you are unlikely to see them or have to deal with them.

  Messages sometimes arrive in "bundles". The beginning of a bundle is marked by a special start-of-bundle control message (for which `C8MessageGetType()` returns C8_MESSAGE_START_BUNDLE) and the end of a bundle is marked with a special end-of-bundle control message (for which `C8MessageGetType()` returns C8_MESSAGE_END_BUNDLE). If you need to handle bundled messages differently from individual messages, you can determinethe start and end of a bundle by recognizing

these messages. If you don't need to deal with bundled messages differently from how you deal with individual messages, then you can simply ignore all messages for which `C8MessageGetType()` returns C8_MESSAGE_START_BUNDLE and C8_MESSAGE_END_BUNDLE.

Parameters:

- msg_ptr: A pointer to the message for which you want to know the type.

Returns: The message type. For a list of the valid message types, see the declaration of C8_MESSAGE_TYPES in the file `c8messages.h`.

- **C8MessageID C8MessageGetID(const C8Message* i_msg) –** Purpose: Given a message, return the ID of that message. Useful when you want to mirror a master window, so that you can identify which row to remove from the mirror when a negative message arrives. The ID of the negative message matches the ID of the original positive message. See "Create Window Statement" in the for more information about master and mirror windows.

  Parameters:

  - i_msg: A pointer to the message for which you want to know the ID.

  Returns: The message ID.

  const C8Schema *C8MessageGetSchema(const C8Message *msg_ptr)

  Purpose: Given a message, return the associated schema. This is useful after `C8AdapterReceiveMessage()` or `C8AdapterReceiveMessageWait()` provides a message that you wish to process depending upon the schema.

  Parameters:

  - msg_ptr: A pointer to the message for which you want to know the schema.

  Returns: A pointer to the schema. Do not change the memory that the pointer points to, and do not `C8Free()` this pointer. The lifetime of the returned pointer is limited to the lifetime of the pointer's owner (in this case, the message).
- **C8Timestamp C8MessageGetMessageTimestamp(const C8Message* msg_ptr) –** Purpose: Given a message, return the associated row timestamp. This is the row timestamp of the message, *not* one of the timestamps in the data columns.

  Parameters:

  - msg_ptr: a pointer to the message for which you want to know the row timestamp.

  Returns: The row timestamp of the message.

  void C8MessageSetMessageTimestamp(C8Message* msg_ptr, C8Timestamp i_timestamp)

  Purpose: Set the row timestamp of the message to i_timestamp. This is the row timestamp of the message and not the timestamp in any columns in the message.

Parameters:

- msg_ptr: The message whose row timestamp you want to set.
- i_timestamp: The timestamp that you want to set the row timestamp to.

Returns: ##########################

- **C8Status C8MessageColumnIsNull(const C8Message* msg_ptr, C8UInt col_ndx, C8Bool *o_res) –** Purpose: For a given message, determines if the column specified by col_ndx is a NULL value or not. Always check for a NULL entry for each column before attempting to read a value from that column, or check the returned status after reading a value from a column.

  Parameters:

  - msg_ptr: A pointer to the message that contains the column.
  - col_ndx: The index number of the desired column within the message. Note that the index is 0-based, not 1-based, and thus valid values range from 0 to N-1 where N is the number of columns in the message.
  - o_res: This variable is set to C8_TRUE if the column is NULL, C8_FALSE otherwise.

  Returns: C8_OK if success, C8_FAIL otherwise.
- **C8Status C8MessageColumnIsNullByName(const C8Message* msg_ptr, const C8CHAR *i_fldnm, C8Bool *o_res) –** Purpose: For a given message, determines if the column whose name is specified in i_fldnm contains a NULL value or not. Always check for a NULL entry for each column before attempting to read a value from that column.

  Parameters:

  - msg_ptr: A pointer to the message that contains the column you want to check.
  - i_fldnm: The name of the column you want to check.
  - o_res : A pointer to a boolean that is set to C8_TRUE if the column has NULL and C8_FALSE otherwise.

  Returns: C8_OK if success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetToNull(C8Message *msg_ptr, C8UInt col_ndx) –** Purpose: Set the column indicated by col_ndx to NULL.

  Parameters:

  - msg_ptr: A pointer to the message containing the column that you want to set to NULL.
  - col_ndx: The index number of the desired column within the message. The index is 0-based, not 1-based, and thus valid values range from 0 to N-1 where N is the number of columns in the message.

  Returns: C8_OK if success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetToNullByName(C8Message *msg_ptr, const C8Char *i_fldnm) –** Purpose: Given a message and column name (i_fldnm) within that message, set that column to NULL.

Parameters:

- msg_ptr: A pointer to the message.
- i_fldnm: The name of the field to set to NULL.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetAsString(const C8Message *i_msg, C8UInt i_ndx, C8Char **o_res) –** Purpose: Given a message and the index of a column in that message, return the value of the column in the form of a string. o_res is set to point to that string. When you are done with the string, free it with `C8Free()`.

  Parameters:

  - i_msg: A pointer to a message.
  - i_ndx: Indicates which column you want to retrieve the value from. Column numbers start at 0, not 1.
  - o_res: A pointer to a location that can store a pointer to the string that is retrieved.

  Returns: C8_OK on success, C8_FAIL otherwise. The string returned by this function must be freed by the user with the `C8Free()` function.
- **C8Status C8MessageColumnGetAsStringByName(const C8Message *i_msg, const C8Char *i_fldnm, C8Char **o_res) –** Purpose: Given a message and the name of a column in that message, return the value of the column in the form of a string. o_res is set to point to that string. When you are done with the string, free it with `C8Free()` .

  Parameters:

  - i_msg: The message from which you want to extract the data.
  - i_fldnm: The name of the field/column to get the data from.
  - o_res: A pointer to a string; the function will set this pointer to point to the string retrieved.

  Returns: C8_OK on success, C8_FAIL otherwise. The string returned by this function must be freed by the user with the `C8Free()` function.
- **C8Status C8MessageColumnGetInt(const C8Message *msg_ptr, C8UInt col_ndx, C8Int *o_res) –** Purpose: Given a message pointer, a column index, and a pointer o_res to a place to store a C8Int, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.

- col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
- o_res: A pointer to a location that can hold the C8Int value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetLong(const C8Message *msg_ptr, C8UInt col_ndx, C8Long *o_res) –** Purpose: Given a message pointer, a column index, and a pointer o_res to a place to store a C8Long, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - o_res: A pointer to a location that can hold the C8Long value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetFloat(const C8Message *msg_ptr, C8UInt col_ndx, C8Float *o_res) –** Purpose: Given a message pointer, a column index, and a pointer o_res to a place to store a C8Float, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - o_res: A pointer to a location that can hold the C8Float value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetBool(const C8Message *msg_ptr, C8UInt col_ndx, C8Bool *o_res) –** Purpose: Given a message pointer, a column index, and a pointer o_res to a place to store a C8Bool, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

- msg_ptr: The message from which you want to extract the data.
- col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
- o_res: A pointer to a location that can hold the C8Bool value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetTimestamp(const C8Message *msg_ptr, C8UInt col_ndx, C8Timestamp *o_res) –** Purpose: Given a message pointer, a column index, and a pointer o_res to a place to store a C8Timestamp, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - o_res: A pointer to a location that can hold the C8Timestamp value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetInterval(const C8Message *msg_ptr, C8UInt col_ndx, C8Interval *o_res) –** Purpose: Given a message pointer, a column index, and a pointer o_res to a place to store a C8Interval, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - o_res: A pointer to a location that can hold the C8Interval value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetString(const C8Message *msg_ptr, C8UInt col_ndx, const C8Char **o_res) –** Purpose: Given a message pointer and column index, set o_res to the C8Char* stored in that message. Do **not** `C8Free` the returned pointer as

this pointer points to a part of the message. The pointer to the string is valid as long as the pointer to the message is valid.

Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()` .

Parameters:

- msg_ptr: The message from which you want to extract the data.
- col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
- o_res: A pointer to a location that can hold a pointer to the string (C8Char *) value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetBlob(const C8Message *msg_ptr, C8UInt col_ndx, const C8Blob **o_res, C8UInt *o_sz) –** Purpose: Given a message pointer and a column index, set o_res to the C8Blob * stored in that message, and set o_sz to the number of bytes in that BLOB. Do **not** C8Free() the returned pointer as this pointer points to a part of the message. The pointer to the BLOB is valid as long as the pointer to the message is valid.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - o_res: A pointer to a location that can hold a pointer to the BLOB value retrieved by the function.
  - o_sz: The function stores the length of the blob in the location pointed to by o_sz.

  Returns: C8_OK on success, C8_FAIL otherwise.

  The retrieved value is a "raw" BLOB (as opposed to a hex string or a base64 string). For an explanation of raw vs. hex string vs. base64 string formats, see *"Data Types and Subroutines for UDFs and In-Process Adapters"* on page 102.
- **C8Status C8MessageColumnGetBlobAsBase64String(const C8Message *msg_ptr, C8UInt col_ndx, const C8Blob **o_res) –** Purpose: Given a message pointer and a column index, return a pointer to a string that contains the base64 representation of the BLOB.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before

retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()` .

Parameters:

- msg_ptr: The message from which you want to extract the data.
- col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
- o_res: A pointer to a location that can hold a pointer to the BLOB value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

The returned value is in a "base64 string" format (as opposed to a hex string or a raw BLOB). For an explanation of raw vs. hex string vs. base64 string formats, see *"Data Types and Subroutines for UDFs and In-Process Adapters"* on page 102.

- **C8Status C8MessageColumnGetXmlAsString(const C8Message *msg_ptr, C8UInt col_ndx, const C8Char **o_res) –** Purpose: Given a message pointer and a column index, set o_res to point to a copy of the XML value represented as a string. When you are done using the XML value, C8Free() the string. Note that this is different from what you do for a C8String or C8Blob.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()` .

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - o_res: A pointer to a location that can hold a pointer to the string value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetIntByName(const C8Message *msg_ptr, const C8Char *i_fldnm, C8Int *o_res) –** Purpose: Given a message pointer, a column name, and a pointer o_res to a place to store a C8Int, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

- msg_ptr: The message from which you want to extract the data.
- i_fldnm: Indicates which column to retrieve data from.
- o_res: A pointer to a location that can hold the C8Int (integer) value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetLongByName(const C8Message *msg_ptr, const C8Char *i_fldnm, C8Long *o_res) –** Purpose: Given a message pointer, a column name, and a pointer o_res to a place to store a C8Long, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - i_fldnm: Indicates which column to retrieve data from.
  - o_res: A pointer to a location that can hold the C8Long value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetFloatByName(const C8Message *msg_ptr, const C8Char *i_fldnm, C8Float *o_res) –** Purpose: Given a message pointer, a column name, and a pointer o_res to a place to store a C8Float, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - i_fldnm: Indicates which column to retrieve data from.
  - o_res: A pointer to a location that can hold the C8Float value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetBoolByName(const C8Message *msg_ptr, const C8Char *i_fldnm, C8Bool *o_res) –** Purpose: Given a message pointer, a column name, and a pointer o_res to a place to store a C8Bool, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before

retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

Parameters:

- msg_ptr: The message from which you want to extract the data.
- i_fldnm: Indicates which column to retrieve data from.
- o_res: A pointer to a location that can hold the C8Bool value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetTimestampByName(const C8Message *msg_ptr, const C8Char *i_fldnm, C8Timestamp *o_res) –** Purpose: Given a message pointer, a column name, and a pointer o_res to a place to store a C8Timestamp, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - i_fldnm: Indicates which column to retrieve data from.
  - o_res: A pointer to a location that can hold the C8Timestamp value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetIntervalByName(const C8Message *msg_ptr, const C8Char *i_fldnm, C8Interval *o_res) –** Purpose: Given a message pointer, a column name, and a pointer o_res to a place to store a C8Interval, put the column's value in the location pointed to by o_res.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - i_fldnm; Indicates which column to retrieve data from.
  - o_res: A pointer to a location that can hold the C8Interval value retrieved by the function.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnGetStringByName(const C8Message *msg_ptr, const C8Char *i_fldnm, const C8Char **o_res) –** Purpose: Given a message pointer and column name, set o_res to the C8Char* stored in that message. Do **not** `C8Free()` the

returned pointer as this pointer points to a part of the message. The pointer to the string is valid as long as the pointer to the tuple is valid.

Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

Parameters:

- msg_ptr: The message from which you want to extract the data.
- i_fldnm: Indicates which column to retrieve data from.
- o_res: A pointer to a location that can hold a pointer to the string (C8Char *) value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnGetBlobByName(const C8Message *msg_ptr, const C8Char *i_fldnm, const C8Blob **o_res, C8UInt *o_sz) –** Purpose: Given a message pointer and a column name, set o_res to the C8Blob * stored in that message, and set o_sz to the number of bytes in that BLOB. Do **not** C8Free() the returned pointer as this pointer points to a part of the message.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - i_fldnm: Indicates which column to retrieve data from.
  - o_res: A pointer to a location that can hold a pointer to the BLOB value retrieved by the function.
  - o_sz: The function stores the length of the blob in the location pointed to by o_sz.

  Returns: C8_OK on success, C8_FAIL otherwise.

  The retrieved blob is a "raw" blob (as opposed to a hex string or a base64 string). For an explanation of raw vs. hex string vs. base64 string formats, see *"Data Types and Subroutines for UDFs and In-Process Adapters"* on page 102.
- **C8Status C8MessageColumnGetXmlAsStringByName(const C8Message *msg_ptr, const C8Char *i_fldnm, const C8Char **o_res) –** Purpose: Given a message pointer and a column name, set o_res to point to a copy of the XML value represented as a string. When you are done using the XML value, C8Free() the string. Note that this is different from what you do for a C8String or C8Blob.

  Prior to using this function on a column for the first time, ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before

retrieving each row's value, ensure that the field contains a value by calling `C8MessageColumnIsNull()`.

Parameters:

- msg_ptr: The message from which you want to extract the data.
- i_fldnm: Indicates which column to retrieve data from.
- o_res: A pointer to a location that can hold a pointer to the string value retrieved by the function.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetInt(C8Message *msg_ptr, C8UInt col_ndx, C8Int value) –** Purpose: Given a message pointer and a column index, set the value in the message to the indicated value. If the column type in the message is not a C8Int, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetLong(C8Message *msg_ptr, C8UInt col_ndx, C8Long value) –** Purpose: Given a message pointer and a column index, set the value in the message to the indicated value. If the column type in the message is not a C8Long, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetFloat(C8Message *msg_ptr, C8UInt col_ndx, C8Float value) –** Purpose: Given a message pointer and a column index, set the value in the message to the indicated value. If the column type in the message is not a C8Float, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.

- col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
- value: The value to store in the message.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetBool(C8Message *msg_ptr, C8UInt col_ndx, C8Bool value) –** Purpose: Given a message pointer and a column index, set the value in the message to the indicated value. If the column type in the message is not a C8Bool, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetTimestamp(C8Message *msg_ptr, C8UInt col_ndx, C8Timestamp value) –** Purpose: Given a message pointer and a column index, set the value in the message to the indicated value. If the column type in the message is not a C8Timestamp, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetInterval(C8Message *msg_ptr, C8UInt col_ndx, C8Interval value) –** Purpose: Given a message pointer and a column index, set the value in the message to the indicated value. If the column type in the message is not a C8Interval, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value : The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetString(C8Message *msg_ptr, C8UInt col_ndx, C8Char* value) –** Purpose: Given a message pointer and a column index, set the column in the message to the indicated value. If the data type of the column is not C8Char *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  If the pointer named "value" was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after calling `C8AdapterSendMessage()` to prevent a memory leak. (The `C8MessageColumnSetString()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.)

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetBlob(C8Message *msg_ptr, C8UInt col_ndx, const C8Blob *i_buf, C8UInt i_sz) –** Purpose: Given a message pointer and a column index, set the column in the message to the indicated value. If the data type of the column is not C8Blob *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  If the i_buf pointer was obtained through `C8Malloc()`, the user must free the pointer via C8Free() after C8AdapterSendMessage() to prevent a memory leak. (The `C8MessageColumnSetBlob()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.)

  Note that the i_buf parameter should contain the original data value - do not convert the data to a string of hexadecimal values first. The function will do this for you. Note that since converting the data to a hexadecimal string will require 2N + 1 bytes (2 hex digits for each byte of the original data, plus a null byte as a string terminator), and since the longest allowable string is 4GB, the longest blob you can put in is 2GB - 1 bytes.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - i_buf: The BLOB value to store in the message. The value is a "raw" BLOB (as opposed to a hex string or a base64 string). For an explanation of raw vs. hex string vs. base64 string formats, see *"Data Types and Subroutines for UDFs and In-Process Adapters"* on page 102.
  - i_sz: The number of bytes in the BLOB.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetBlobFromBase64String(C8Message *msg_ptr, C8UInt col_ndx, const C8Char *i_base64_string) –** Purpose: Given a message pointer and a column index, set the column in the message to the indicated value. If the data type of the column is not C8Blob *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  If the i_base64_string pointer was obtained through `C8Malloc()`, the user must free the pointer via C8Free() after C8AdapterSendMessage() to prevent a memory leak. (The `C8MessageColumnSetBlob()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.)

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The BLOB value to store in the message. The value is a "raw" BLOB (as opposed to a hex string or a base64 string). For an explanation of raw vs. hex string vs. base64 string formats, see *"Data Types and Subroutines for UDFs and In-Process Adapters"* on page 102.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetXmlFromString(C8Message *msg_ptr, C8UInt col_ndx, C8Char* value) –** Purpose: Given a message pointer and a column index, set the column in the message to the indicated value. If the data type of the column is not a C8Char *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after `C8AdapterSendMessage()` to prevent a memory leak. (The `C8MessageColumnSetXmlFromString()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.)

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetIntByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Int value) –** Purpose: Given a message pointer and a column name, set the value in the message to the indicated value. If the schema type in the message is not a C8Int, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

Parameters:

- msg_ptr: The message from which you want to extract the data.
- col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
- value: The value to store in the message.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetLongByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Long value) –** Purpose: Given a message pointer and a column name, set the value in the message to the indicated value. If the schema type in the message is not a C8Long, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetFloatByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Float value) –** Purpose: Given a message pointer and a column name, set the value in the message to the indicated value. If the schema type in the message is not a C8Float, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetBoolByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Bool value) –** Purpose: Given a message pointer and a column name, set the value in the message to the indicated value. If the schema type in the message is not a C8Bool, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.

- value: The value to store in the message.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetTimestampByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Timestamp value) –** Purpose: Given a message pointer and a column name, set the value in the message to the indicated value. If the schema type in the message is not a C8Timestamp, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx : Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetIntervalByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Interval value) –** Purpose: Given a message pointer and a column name, set the value in the message to the indicated value. If the schema type in the message is not a C8Interval, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8MessageColumnSetStringByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Char* value) –** Purpose: Given a message pointer and a column name, set the column in the message to the indicated value. If the data type of the column is not a C8Char *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after `C8AdapterSendMessage()` to prevent a memory leak. (The `C8MessageColumnSetStringByName()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.)

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.

- value: The value to store in the message.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetBlobByName(C8Message *msg_ptr, const C8Char *i_fldnm, const C8Blob * i_buf, C8UInt i_sz) –** Purpose: Given a message pointer and a column name, set the column in the message to the indicated value. If the data type of the column is not a C8Blob *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error message is logged.

  If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after `C8AdapterSendMessage()` to prevent a memory leak. (The `C8MessageColumnSetBlobByName()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.)

  Note that the i_buf parameter should contain the original data value -- do not convert the data to a string of hexadecimal values first -- the function will do that for you. Note that since converting the data to a hexadecimal string will require 2N + 1 bytes (2 hex digits for each byte of the original data, plus a null byte as a string terminator), and since the longest allowable string is 4GB, the longest blob you can put in is 2GB - 1 bytes.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value - the value to store in the message. The value is a "raw" BLOB (as opposed to a hex string or a base64 string). For an explanation of raw vs. hex string vs. base64 string formats, see *"Data Types and Subroutines for UDFs and In-Process Adapters"* on page 102.

  Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8MessageColumnSetXmlFromStringByName(C8Message *msg_ptr, const C8Char *i_fldnm, C8Char* value) –** Purpose: Given a message pointer and a column name, set the column in the message to the indicated value. If the data type of the column is not a C8Char *, the results are undefined and an error is logged. If the col_ndx is out of range, the set is ignored and an error is message logged.

  If the value pointer was obtained through `C8Malloc()`, the user must free the pointer via C8Free() after `C8AdapterSendMessage()` to prevent a memory leak. The `C8MessageSetXmlFromStringByName()` routine will make a copy of the value, so freeing that value does not cause the server to lose access to the information.

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.

- value: The value to store in the message.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Char* C8MessageSerialize(const C8Message *msg_ptr, enum C8_MESSAGE_FORMAT fmt) –** Purpose: A serialized form of msg_ptr is returned as formatted by fmt. Currently the only format supported is C8_MESSAGE_XML. You are responsible for freeing the returned string via C8Free().

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Message* C8MessageDeserialize(const C8Char* string, enum C8_MESSAGE_FORMAT fmt) –** Purpose: The input string of format fmt is converted to a message. Currently the only format supported is C8_MESSAGE_XML. You are responsible for freeing the returned string via C8Free().

  Parameters:

  - msg_ptr: The message from which you want to extract the data.
  - col_ndx: Indicates which column to retrieve data from. Column numbers start from 0, not 1.
  - value: The value to store in the message.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8SizeType C8MessageBatchGetCount(C8MessageBatch* batch) –** Purpose: Determines the number of messages in a batch.

  Parameters:

  - batch: The batch you want a count of.

  Returns: The number of messages in the batch.
- **C8Message* C8MessageBatchPopMessage(C8MessageBatch* `batch`) –** Purpose: Extracts the next message from a batch.

  Parameters:

  - batch: The batch from which you want to extract a message.

  Returns: A pointer to a message or NULL if there are no messages in the batch. Use `C8MessageDestroy` to destroy each message when you are finished with it.
- **const C8Char* C8MessageBatchGetId(C8MessageBatch* batch) –** Purpose: Retrieves the ID of this batch of messages.

  Parameters:

- batch: The batch for which you want the ID.

Returns: A NULL-terminated string containing the batch ID.

- **void C8MessageBatchDestroy(C8MessageBatch* batch) –** Purpose: Destroys a batch, including all the messages, and deallocates the associated memory.

  Parameters:

  - batch: The batch you want to destroy.

  Returns: Nothing.

Creating an Out-of-process Adapter in C/C++

If required, you can write your own out-of-process adapter in the C/C++ programming language. You can write either an input adapter or an output adapter with C/C++.

Sybase CEP provides a set of functions that allow you to connect to a stream, read or write data, and detach from the stream. These functions are referred to as the Sybase CEP C Adapter library.

Sybase CEP provides:

- A library for resolving a CCL stream URI.
- A library for transferring data via a network.
- Documentation.

Sybase CEP provides a sample out-of-process input adapter and a sample out-of-process output adapter:

- example_input_adapter.c.
- example_output_adapter.c.

Both examples are located in the same directory. On Microsoft Windows, the default location in the directory is:

```
C:\Program Files\SybaseC8\Server\sdk\c\examples
```

On UNIX-like operating systems the default directory is:

```
/home/<userid>/sybasec8/server/sdk/c/examples
```

C/C++ API Interface adapters

An overview of the behaviour of the C/C++ API interface adapters, including their functions and parameters.

As we mentioned earlier, a typical out-of-process adapter performs the following tasks:

1. Acquiring an address (URI) that will uniquely identify a specific stream and telling the communication layer (provided by) how to find that stream.
2. Opening a connection to a tream.
3. Reading (or writing) the desired data. Typically the read (or write) operation is in a loop.
4. Closing the connection (if the adapter is not going to run indefinitely).

If the adapter is reading from the stream, then the adapter may need to wait until the next message arrives. There are three possible ways that the adapter can wait for the next message:

- The adapter can "poll" until a message arrives.
- The adapter can make a blocking call that will wait until a message arrives.
- The adapter can register a callback function that will be called when a message arrives.

To poll, the adapter should call the `C8SubscriberGetNextMessage()` function and specify a maximum amount of time to wait. If a message does not arrive within the specified time, `C8SubscriberGetNextMessage()` will return control to the caller, which may do some work (or yield the processor) and then continue polling later by calling `C8SubscriberGetNextMessage()` again. Passing 0 as the wait time returns control immediately, without waiting.

To register a callback function, use the `C8SubscriberSetCallback()` function.

Below we describe all the functions in the API, including the functions that allow users to attach to streams, read or write streams, and detach from streams. (Acquiring a stream's address is done outside the API, and will be explained later.) The header file `c8adapter.h` describes the interface.

Remember to call `C8ClientSDKInitialize()` before calling any functions in an out-of-process adapter.

- **C8Publisher *C8PublisherCreate(const C8Char *i_uri) –** Purpose: Create a publisher to the given URI for publishing messages. The publisher object returned by this function is used later when publishing messages.

  Parameters:

  - i_uri: The URI of the stream you want to publish to.

  Returns: A pointer to a C8Publisher object (or NULL if error). Publishers returned by this function must be freed with `C8PublisherDestroy()`.
- **C8Publisher *C8PublisherCreateA(const C8Char *i_uri, const C8UserCredentials *i_credentials) –** Purpose: Like `C8PublisherCreate()`, this creates a publisher to the given stream URI, but this function requires user authentication. The returned publisher object is used later when sending (publishing) messages.

  Parameters:

  - i_uri: The URI of the stream you want to publish to.

- i_credentials: A pointer to a structure containing your credentials (username and password).

Returns: A pointer to a C8Publisher object (or NULL if error). Publishers returned by this function must be freed with `C8PublisherDestroy()`.

- **C8Publisher *C8PublisherCreateGD(const C8Char *i_uri, const C8UserCredentials *i_credentials, const C8Char *i_session_id, C8Interval i_timeout –** Purpose: Creates a Guaranteed Delivery publisher to the given stream URI (see *Implementing Guaranteed Processing* on page 89 for information about Guaranteed Delivery) and establishes (or re-establishes) a session with CEP Server. The returned publisher object is used later when sending (publishing) messages.

  Parameters:

  - i_uri: The URI of the stream you want to publish to.
  - i_credentials: (Optional). A pointer to a structure containing your credentials (user name and password).
  - i_session_id: The unique session ID for Guaranteed Delivery. If you are reconnecting after a failure, be sure to use the same session ID.
  - i_timeout: How long to wait to establish a session before timing out.

  Returns: A pointer to a C8Publisher object (or NULL if error). Publishers returned by this function must be freed with `C8PublisherDestroy()`.
- **void C8PublisherDestroy(C8Publisher *i_pub) –** Purpose: Frees a C8Publisher object and any associated schema objects.

  Parameters:

  - i_pub: A pointer to the publisher object to be destroyed (de-allocated).

  Returns:
- **C8Bool C8PublisherIsConnected(C8Publisher * i_pub) –** Purpose: Determines whether or not a publisher is connected and ready to send messages.

  Parameters:

  - i_pub: The publisher.

  Returns: C8_TRUE if connected, C8_FALSE otherwise.
- **const C8Schema * C8PublisherGetSchema(const C8Publisher *i_pub) –** Purpose: Get the schema of the stream that is being published to.

  Parameters:

  - i_pub: The publisher for which you want to get the schema.

  Returns: A pointer to a schema. The returned schema remains valid while the publisher remains valid. you should not C8Free() the schema; the schema will be de-allocated when the publisher is de-allocated.

- **C8Status C8PublisherSendMessage(C8Publisher *i_pub, C8Message *i_msg) –** Purpose: Publishes a message to the stream whose URI was specified when the publisher *i_pub was created by calling C8PublisherCreate().

  Once a message is published, modifying the message is not allowed. However, you may deallocate a message with C8MessageDestroy() at any time. If the publisher is a Guaranteed Publisher, the message is not actually sent to the stream until you call C8PublisherCommit.

  Parameters:

  - i_pub: The publisher for which you want to get the schema.
  - i_msg: The message that you want to publish.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Status C8PublisherSendMessages(C8Publisher *i_pub, C8Message **i_msg, C8UInt i_cnt) –** Purpose: Publishes an array of messages to a stream. If the publisher is a Guaranteed Publisher, the messages are not actually sent to the stream until you call C8PublisherCommit.

  Parameters:

  - i_pub: The publisher that specifies which stream to publish to.
  - i_msg: A pointer to an array of messages to be sent.
  - i_cnt: The number of messages in the array.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Status *C8PublisherCommit(C8Publisher *i_pub, const C8Char *i_batch_id) –** Purpose: Send all messages that have been queued with C8PublisherSendMessage or C8PublisherSendMessages as a batch. Used with a Guaranteed Delivery publisher.

  Parameters:

  - i_pub: The Guaranteed Delivery publisher that specifies which stream to publish to.
  - i_batch_id: The ID of this batch. See *Implementing Guaranteed Processing* on page 89 for information about Guaranteed Delivery.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Status * C8PublisherSendMessageBatch(C8Publisher * i_pub, C8Message** i_msgs, C8UInt i_cnt, const C8Char * i_batch_id) –** Purpose: Send a batch of messages. Used with a Guaranteed Delivery publisher.

  Parameters:

  - i_pub: The Guaranteed Delivery publisher that specifies which stream to publish to.
  - i_msgs: An array of messages
  - i_cnt: The number of messages in i_msgs

- i_batch_id: The unique ID of this batch. See *Implementing Guaranteed Processing* on page 89 for information about Guaranteed Delivery.

Returns: C8_OK if successful, C8_FAIL otherwise.

- **const C8Char *C8PublisherGetLastBatchId(C8Publisher *i_pub) –** Purpose: Retrieves the ID of the last batch of messages received from a Guaranteed Delivery publisher.

  Parameters:

  - i_pub: The Guaranteed Delivery publisher.

  Returns: A constant pointer to the ID of the last batch processed by the server, or NULL if no batch has been processed since the last restart with clean slate.
- **C8Subscriber *C8SubscriberCreate(const C8Char *i_uri) –** Purpose: Creates a subscriber to a given stream.

  Parameters:

  - i_uri: The URI of the stream to subscribe to.

  Returns: Pointer to a subscriber object, or NULL if an error occurs. Subscribers should be freed with `C8SubscriberDestroy()`.
- **C8Subscriber *C8SubscriberCreateA(const C8Char *i_uri, const C8UserCredentials *i_credentials) –** Purpose: Creates a subscriber to a given stream that requires user authentication.

  Parameters:

  - i_uri: The URI of the stream to subscribe to.
  - i_credentials: A pointer to a structure containing your credentials (username and password).

  Returns: A pointer to a subscriber object or NULL if an error occurs. Subscribers should be freed with `C8SubscriberDestroy()`.
- **C8Subscriber *C8SubscriberCreateGD(const C8Char *i_uri, const C8UserCredentials *i_credentials, const C8Char * i_session_id, const C8Char * i_last_batch_id, C8Interval i_timeout) –** Purpose: Creates a Guaranteed Delivery subscriber to a given stream and establishes (or re-establishes) a session with the server.

  Parameters:

  - i_uri: The URI of the stream to subscribe to.
  - i_credentials: A pointer to a structure containing your credentials (username and password). (Optional.)
  - i_session_id : The unique session ID for Guaranteed Delivery. If you are reconnecting after a failure, be sure to use the same session ID.

- i_last_batch_id: The ID of the last batch processed by this adapter. The server should begin sending messages with the batch following the one specified here.
- i_timeout: How long to wait to establish a session before timing out.

See *Implementing Guaranteed Processing* on page 89 for information about Guaranteed Delivery.

Returns: A pointer to a subscriber object or NULL if an error occurs. Subscribers should be freed with `C8SubscriberDestroy()`.

- **C8Message * C8SubscriberGetNextMessage(C8Subscriber *i_sub, C8Interval i_to) –** Purpose: Gets the next pending message from the subscriber. If this is a Guaranteed Delivery subscriber, call `C8SubscriberGetNextBatchID()` to determine if this message begins a new batch of messages.

  Parameters:

  - i_sub: The subscriber object from which to get the next message.
  - i_to: Indicates the maximum amount of time (in microseconds) to wait for the next message.

  Returns: The next pending message, or NULL if no messages are pending. In case of a timeout, the call to `C8ErrorGetCode()` returns C8_ERR_TIMEOUT; a different error code is returned if there was an actual error. The message must be deleted with `C8MessageDestroy()`.

  Warning! Note that whether or not the timeout was reached, this function may return NULL to indicate that there were no messages, so you must always check the return value from this function - do not assume that you will always get a message.

- **C8Status C8SubscriberGetNextBatch(C8Subscriber * i_sub, C8Message** o_msgs, C8UInt* io_cnt, const C8Char** o_batch_id, C8Interval i_to) –** Purpose: Get the next batch of messages from the guaranteed delivery subscriber.

  Parameters:

  - i_sub: The Guaranteed Delivery subscriber object from which to get the next batch of messages.
  - o_msgs: A pointer to an array to hold the returned messages.
  - io_cnt: The number of messages that will fit into 0_msgs (input) or the number of messages in o_msgs (output). If the entire batch of messages does not fit in the allocated array, subsequent calls to this method will retrieve the rest of the batch.
  - o_batch_id: The ID of this batch of messages.
  - i_to : Indicates the maximum amount of time (in microseconds) to wait for the batch.

  Returns: C8_OK if successful, C8_FAIL otherwise. The messages must be deleted with `C8MessageDestroy()`.

- **const C8Char *C8SubscriberGetLastBatchId(C8Subscriber *i_sub) –** Purpose: Retrieves the ID of the last batch of messages sent by the server to a Guaranteed Delivery subscriber.

Parameters:

- i_sub: The Guaranteed Delivery subscriber.

Returns: A constant pointer the ID of the last batch sent.

- **void C8SubscriberSetCallback(C8Subscriber *i_sub, C8SubscriberCallbackFn i_cb, void *i_cb_prm) –** Purpose: Sets a callback function that will be called when a new message is available on the stream that has been subscribed to.

  The callback function will be called by the adapter library code (which is linked into your out-of-process adapter) and therefore will be running in a thread that is running the library code. Thus, the callback function should be as short as possible and require minimum locking. The callback function should not itself process the newly-arrived message. In many cases, the callback function may simply set a flag and then return, leaving other parts of your code to check the flag, retrieve the message, and then process the message when their thread executes.

  Warning! In some cases the callback routine may be called even if there is no message. When you call C8SubscriberGetNextMessage() (which you should NOT call inside the callback routine itself), be sure to check that C8SubscriberGetNextMessage() returned a valid C8Message pointer, not NULL.

  Parameters:

  - i_sub: The subscription whose stream you want to be notified about.
  - i_cb : The new callback function for this subscriber.
  - i_db_prm: The parameter with which to call the callback.

  Returns: Nothing.

- **void C8SubscriberDestroy(C8Subscriber *i_sub) –** Purpose: Frees the subscriber.

  Parameters:

  - i_sub: The subscriber that you want to free or de-allocate.

  Returns: Nothing.

- **C8Bool C8SubscriberIsConnected(C8Subscriber * i_sub) –** Purpose: Determines whether or not a subscriber is connected and ready to receive messages.

  Parameters:

  - i_sub: The subscriber.

  Returns: C8_TRUE if connected, C8_FALSE otherwise.

- **const C8Schema *C8SubscriberGetSchema(const C8Subscriber *i_sub) –** Purpose: Get the schema of the stream subscribed to.

  Parameters:

  - i_sub: The subscriber specifies which stream you want to know the schema of.

Returns: The schema of the stream you've subscribed to or NULL if an error occurs. The returned schema remains valid while the subscriber remains valid.

- **C8Char * C8ResolveUri(const C8Char * i_uri); –** Purpose: Resolves a CCL URI into an HTTP/HTTPS URI.

  Parameters:

  - i_uri: The URI to resolve.

  Returns: The HTTP URL or NULL if an error occurs. The pointer returned must be freed with C8Free().
- **C8Char * C8ResolveUriA(const C8Char * i_uri, const C8UserCredentials *i_credentials); –** Purpose: Resolves a CCL URI into an HTTP/HTTPS URI, requiring that you pass appropriate user authentication credentials before doing so.

  Parameters:

  - i_uri: The URI to resolve.
  - i_credentials: A pointer to a structure containing your credentials (user name and password).

  Returns: The HTTP URL. This returns NULL if an error occurs. The pointer returned must be freed with C8Free().
- **C8Char ** C8ResolveClusterUri(const C8Char * i_uri); –** Purpose: Resolves a CCL URI into one or more HTTP/HTTPS URIs.

  Parameters:

  - i_uri: The URI to resolve.

  Returns: A pointer to a NULL-terminated array of string pointers, or NULL if an error occurs. The returned array pointer and all of it's contained pointers must be freed with `C8Free()`.
- **C8Char ** C8ResolveClusterUri(const C8Char * i_uri, const C8UserCredentials *i_credentials); –** Purpose: Resolves a CCL URI into one or more HTTP/HTTPS URIs, requiring that you pass appropriate user authentication credentials before doing so.

  Parameters:

  - i_uri: The URI to resolve.
  - i_credentials: A pointer to a structure containing your credentials (user name and password).

  Returns: A pointer to a NULL-terminated array of string pointers, or NULL if an error occurs. The returned array pointer and all of it's contained pointers must be freed with C8Free().
- **C8Char * C8UriAppendQueryString(const C8Char* i_base_uri, const C8Char* i_query_string); –** Purpose: Appends a query string to the passed-in URI. Useful if you want to specify filters when subscribing to a stream.

Parameters:

- i_base_uri: The URI to extend.
- i_query_string: The URL-encoded string to append to i_base_uri.

Returns: The URI in the form base_uri?query_string. This returns NULL if an error occurs. The pointer returned must be freed with `C8Free()`.

- **C8Char * C8UriAppendQueryStringParameter(const C8Char* i_base_uri, const C8Char *i_key, const C8Char* i_value); –** Purpose: Appends query parameters to the passed-in URI. Useful if you want to specify filters when subscribing to a stream.

  Parameters:

  - i_base_uri: The URI to extend.
  - i_key: The unencoded key.
  - i_value: The unencoded value.

  Returns: The URL-encoded URI in the form base_uri?key=value. This returns NULL if an error occurs. The pointer returned must be freed with `C8Free()`.

Creating a Sample Input Adapter

An overview of connecting a simple adapter for writing to Sybase CEP Server.

To create and run our demo, we use example_input_adapter.c: This file is a very basic input adapter that attaches to a stream, writes data to the stream, and then detaches from the stream.

This example may be used as a skeleton for other out-of-process adapters. This file is in:

```
server/sdk/c/examples
```

For example, on Microsoft Windows, the directory is typically

```
C:\Program Files\SybaseC8\Server\sdk\c\examples
```

On UNIX-like operating systems, the directory is typically

```
/home/<username>/sybasec8/server/sdk/c/examples
```

Sample Input Adapter

Describes the example input adapter, which shows how to write data to Sybase CEP Server.

This example generates its own data; when you write your own adapter, you will probably read the data from a data source.

After the normal declarations and command line argument parsing, the input stream is opened.

Acquiring the Address (URI) of a Stream

Acquire the URI address of the stream in order to pass it to the function that opens a connection to the stream.

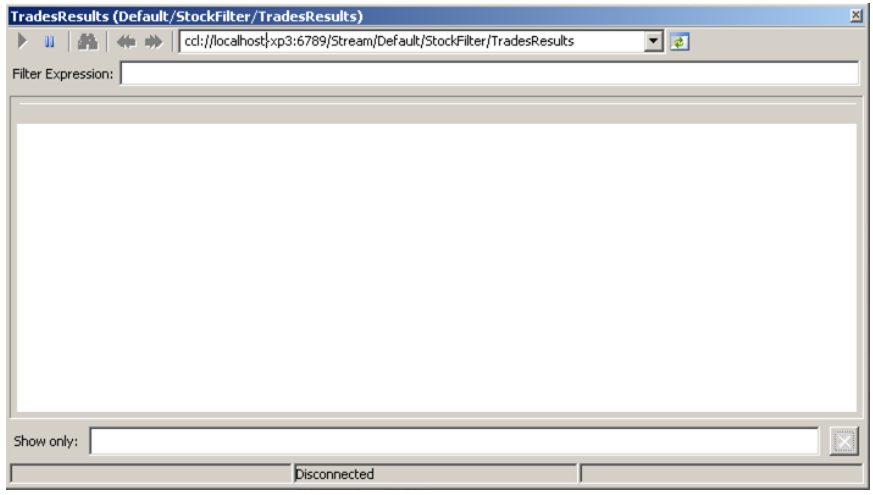
To get the URI of a particular stream, you must:

1. Create the stream in Sybase CEP Studio.
2. Open a stream viewer by right-clicking on the stream and clicking the "view" command.
3. Read the URI shown in the title of the stream viewer window.

The URI will look similar to the one shown below:

```
ccl://localhost:6789/Stream/Default/PassThrough/InTrades
```

You can see this URI at the top of the figure below:



The URI of the stream does not change unless the stream name itself changes or unless the query module and stream are run on a different host computer.

For more information about URIs, see *Stream URI* on page 8.

Compiling and Linking the Example

This section explains how to compile and link the example code on Microsoft Windows and UNIX-like operating systems. In Microsoft Windows, the example can be compiled and linked either from inside Visual Studio or from the command line. In UNIX-like operating systems, the example can be compiled and linked from within a C compiler.

Compiling Using Visual Studio in Microsoft Windows

This section explains how to compile and link on Microsoft Windows from inside Visual Studio. These instructions assume that you are using Microsoft Visual Studio 2005.

1. Start Microsoft Visual Studio.
2. Create a project file by going to the menu and clicking on File -> New -> Project.
   a) Click on the [+] to expand "Visual C++ " Project Types.
   b) Click on "Win32 Console Application".
   c) In the "Name:" field, fill in the name that you'd like to use for your project.
   d) In the "Location:" field, browse and specify the directory where you want project to be stored.
   e) Click OK.
   f) The next window to appear is the "Win32 Application Wizard" window. Click on "Finish".
3. Microsoft Visual Studio creates a simple C++ (.cpp) file to use as a starting point. Remove all the contents of this file and then insert your own code for the adapter.

   Make sure your file includes the following:

   ```
   #include <nspr.h>
       #include "c8adapter.h"
       #include "c8client.h"
       #include "c8conversions.h"
       #include "c8messages.h"
   ```
4. If you have other C-language source files that you need, add them to the project
5. Update several settings that are available in the "Property Pages" for this project:
   a) Update the list of directories to search for #include files:
      1. Go to the menu, click on "Project" and then on "MySample Properties". This opens a new window named "MySample Property Pages" or similar.
      2. In the left-hand pane of this window, click on "Configuration Properties", then on , and then on "General". The right-hand pane should now show a list of settings that you can modify.
      3. Click in the field to the right of "Additional Include Directories" and add the directories listed below. On Microsoft Windows, these directories are typically:

      ```
      C:\Program Files\SybaseC8\Server\sdk\c\include\
              C:\Program Files\SybaseC8\Server\sdk\c\include\nspr
      ```

      On Microsoft Windows, the directories are typically:

      ```
      C:\Program Files (x86)\SybaseC8\Server\sdk\c\include
              C:\Program Files (x86)\SybaseC8\Server\sdk\c\include
      \nspr
      ```

      You can add other directories if necessary for your adapter:

      1. Click in the "Additional Include Directories" field, and paste in the first path.
      2. Click on the ellipsis ("..."), click on the folder icon (the tool tip will refer to this as "new line"), and then enter the next path.

Repeat step "b" as many times as necessary to add the remaining required paths.

b) Turn off precompiled headers. Go to the left-hand pane in the "Property Pages" window, click on "" and then on "Precompiled headers", then click on "Create/Use Precompiled Header" and set it to "Not Using Precompiled Headers".

c) Add the library files. Go to the left-hand pane of the "Property Pages" window, click on "Linker" and then on "General".

   In the field to the right of "Additional Library Directories", add the directory that contains the library.

   On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk\c\lib
```

   On Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\SybaseC8\Server\sdk\c\lib
```

d) Add a dependency on the `c8_sdk_client_lib.lib` file. To do this, go to the left-hand pane of the "Property Pages" window, click on "Linker" and then on "Input".

   In the field to the right of "Additional Dependencies", enter

   `c8_sdk_client_lib.lib nspr4.lib`

   (Note that you do not need to enter the complete path of either of these libraries; entering the file names is sufficient.)

   If you are using Microsoft's Visual C development and environment and you'd like to double check that you haven't skipped a step, you can look at the Command Line for the compiler and the Command Line for the Linker. These show the command-line parameters passed from Microsoft's GUI IDE to the command-line compiler and linker.

   To view the command line for the compiler, go to the left-hand pane of the Property Pages window, click on "" and then click on Command Line. The command line should look similar to the following:

```
/Od /I "C:\Program Files\SybaseC8\Server\sdk\c\include\nspr"
    /I "C:\Program Files\SybaseC8\Server\sdk\c\include\\"
    /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D "_UNICODE"
    /D "UNICODE" /Gm /EHsc /RTC1 /MDd /Fo"Debug\\"
    /Fd"Debug\vc80.pdb" /W3 /nologo /c /Wp64 /ZI
    /TP /errorReport:prompt
```

   If you set the warning level to a value other than 3, then the "/W3" will be different.

   The command line may or may not include

```
/D "_DEBUG"
```

If the command line includes this, you may only be able to use the .DLL on a computer that has the debug version of the C runtime library. For more information, see the *Troubleshooting* on page 351 section.

To view the command line for the linker, go to the left-hand pane of the Property Pages window, click on "Linker" and then click on "Command Line". The command line should look similar to the following:

```
/OUT:"C:\c8test\E3\SDKC\TESTAdapter1.2\TESTAdapter1.2\Debug
\TESTAdapter1.2.exe"
  /INCREMENTAL /NOLOGO /LIBPATH:"C:\Program Files
\SybaseC8\Server\sdk\c\lib"
  /MANIFEST /MANIFESTFILE:"Debug
\TESTAdapter1.2.exe.intermediate.manifest"
  /DEBUG
  /PDB:"c:\c8test\e3\sdkc\testadapter1.2\testadapter1.2\debug
\TESTAdapter1.2.pdb"
  /SUBSYSTEM:CONSOLE /MACHINE:X86
  /ERRORREPORT:PROMPT c8_sdk_client_lib.lib nspr4.lib
kernel32.lib user32.lib
    gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib
    oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
```

Now that you have entered all the project properties, click the "OK" button on the "Property Pages" window.

At this point, you are ready to compile.

6. To compile, use the appropriate option on the "Build" menu, for example, "Build MySample".
7. To execute the file that you have created, you must ensure that your PATH environment variable includes the directories that contain the .dll files that the adapter needs. Typically, these directories are:

```
C:\Program Files\SybaseC8\Server\bin
    C:\Program Files\SybaseC8\Server\sdk\c\lib
```

Compiling Using the Command Line in Microsoft WIndows

This section explains how to compile and link the example code on Microsoft Windows by using the command line. These instructions assume that you are using the compiler that comes with Microsoft Visual Studio 2005.

1. Configure your C compiler so that you can compile and link at least a simple C program (for example, a "hello world" program). A sample script that sets environment variables is available later in this section.

   To ensure that you can access the MS Visual Studio 2005 resources such as library files you will typically need to set the environment variables PATH, INCLUDE, and LIB

appropriately. Set these variables by running the vcvars32.bat script provided by Microsoft with Visual Studio:

```
C:\Program Files\Microsoft Visual Studio 8\VC\bin\vcvars32.bat
```

2. Update your INCLUDE environment variable to include the following directories:

```
C:\Program Files\SybaseC8\Server\sdk\c\include\
    C:\Program Files\SybaseC8\Server\sdk\c\include\nspr
```

3. Update your LIB environment variable to include the following directory:

```
C:\Program Files\SybaseC8\Server\sdk\c\lib
```

This allows the compiler/linker to find the required .lib files when linking.

To do this, execute the following command:

```
set LIB=%LIB%;C:\Program Files\SybaseC8\Server\sdk\c\lib
```

4. When you execute the program, included the following files:

```
C:\Program Files\SybaseC8\Server\sdk\c\lib
    C:\Program Files\sybasec8\server\bin
```

This allows the program to find the .dll (Dynamic Link Library) files or .so (Shared Object) files at runtime.

To do this, execute the following commands:

```
set PATH=%PATH%;C:\Program Files\SybaseC8\Server\sdk\c\lib
    set PATH=%PATH%;C:\Program Files\SybaseC8\Server\bin
```

5. On Microsoft Windows, execute "cl" (Compile and Link) commands like those shown below:

```
rem Compile and link the C Adapter SDK example
    rem program, which uses the Adapter API.   cl
example_input_adapter.c c8_sdk_client_lib.lib nspr4.lib
```

Below is a sample Microsoft Windows .bat file to configure the environment for compiling and running the C Adapter SDK example with the Microsoft Visual Studio .NET 2005 development environment.

```
@echo off
rem
-----------------------------------------------------------------
--
rem PURPOSE: The purpose of this file is to configure the
environment
rem     so that we can compile stream adapters, which require that
rem     we have the NSPR (Netscape Portable Runtime) files, which
rem      include .h and library files that we need.
```

```
rem     This assumes that you have installed
rem     Microsoft Visual Studio .NET 2005 to the directory
rem        C:\Program Files\Microsoft Visual Studio 8
rem     and have already run the vcvars32.bat script.
rem
-----------------------------------------------------------------
--
rem --- Set the env vars to enable us to find the C8 .h files
rem --- and the NSPR .h files.
set INCLUDE=C:\Program Files\SybaseC8\Server\sdk\c\include;
%INCLUDE%
set INCLUDE=C:\Program Files\SybaseC8\Server\sdk\c\include\nspr;
%INCLUDE%
rem --- Set the env vars to enable us to find the Sybase library
files
set LIB=C:\Program Files\SybaseC8\Server\sdk\c\lib\;%LIB%
rem Needed for SSL.
rem NOTE TO READER: The following should all be on one
rem   line, not split across lines.
set INCLUDE=C:\Program Files\SybaseC8\Server\sdk\c\include\nss
\public;%INCLUDE%
rem --- Make sure that the PATH contains the .DLL files.
set PATH=%PATH%;c:\Program Files\SybaseC8\Server\bin
set PATH=%PATH%;c:\Program Files\SybaseC8\Server\sdk\c\lib
```

Compiling and Linking on UNIX-like Operating Systems

This section explains how to compile and link the example code on UNIX-like operating systems.

1. Configure your C compiler (for example, gcc) so that you can compile and link at least a simple C program (for example, a "hello world" program).

   Set the environment variables PATH, INCLUDE, and LIB appropriately.
2. Update your LIB environment variable to include the proper library directory. If you have the Sybase CEP Engine installed under the directory "/home/<userid>" then the paths will be:

   ```
   /home/<userid>/sybasec8/server/sdk/c/lib
   ```

   To update your LiBenvrionment variable, execute the following command:

   export LIB=${LIB}:/home/<userid>/sybasec8/server/sdk/c/lib

   This allows the compiler/linker to find the required .lib files when linking.
3. Make sure that your PATH environment variable includes the cirrecr by executing the following commands:

   ```
   export PATH=${PATH}:/home/<userid>/sybasec8/server/sdk/c/lib
       export PATH=${PATH}:/home/<userid>/sybasec8/server/bin
   ```

   This allows the program to find the .so (Shared Object) files at runtime.

4. Compile and link:

```
# Compile the bulk of the code and link all the
    # pieces together. Note that the command below
    # should all be on one line, even if
    # it is displayed as multiple lines.
    gcc -Wl,--allow-shlib-undefined -I${INCLUDE1}
   -I${INCLUDE2} -L${LIB1} -L${LIB2} myAdapter.c
   ${LIB1}/libc8_sdk_client_lib.so
```

Below is a sample shell script to configure the environment for compiling and running the C Adapter SDK example. Customize this file to specify the userid. If you do not have the Sybase CEP Engine installed in "/home/<userid>" you will need to make additional customizations. (The example assumes that you are using gcc on linux).

```
# PURPOSE:
#    Set up the environment so that we can compile an
#    out-of-process adapter.
echo "Don't forget to 'source' this script"
echo "(for example, with the '.' command) "
echo "rather than simply run it."
# Include the standard  C/C++ SDK files.
# We'll use this environment variable later when we compile.
export INCLUDE1=/home/<userid>/sybasec8/server/sdk/c/include
# Include additional files needed for out-of-process adapters.
# We'll use this environment variable later when we compile.
export INCLUDE2=/home/<userid>/sybasec8/server/sdk/c/include/nspr
export LIB1=/home/<userid>/sybasec8/server/sdk/c/lib
export LIB2=/home/<userid>/sybasec8/server/bin
export PATH=$PATH:$LIB1:$LIB2
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$LIB1:$LIB2
```

Below is a sample shell script to compile and link the adapter. Customize this file to specify the program name; we used "myAdapter" below. (The example below assumes that you are using gcc on linux).

```
# PURPOSE:
#    Compile and link the out-of-process adapter.
#    This script is for UNIX-like operating systems.
echo "This assumes that you already ran set_adapter_env.bat, "
echo "which sets the INCLUDE and LIB env vars."
# Compile the bulk of the code and link all the pieces together.
# Note that the command below should all be on one line, even if
# it is displayed as multiple lines.
gcc -Wl,--allow-shlib-undefined -I$INCLUDE1 -I$INCLUDE2 -L$
{LIB1}
  -L${LIB2} myAdapter.c ${LIB1}/libc8_sdk_client_lib.so
# Give the executable file whatever name you want.
mv a.out myAdapter
```

The "--allow-shlib-undefined" command tells the gcc compiler to proceed even if it can't find all the library files it needs. The required library files are in the PATH or the LD_LIBRARY_PATH and will be resolved at runtime.

Executing the Out-of-process Input Adapter

This section describes how to execute the out-of-process input adapter.

These instructions assume that you have already installed the Sybase CEP Server and Sybase CEP Studio, and that you have compiled the programs according to the instructions in the previous section.

1. Start the server and let it fully initialize.
2. Start Sybase CEP Studio.
3. Start Sybase CEP Studio, connect to the Default workspace, and load the PassThrough example, located in the SybaseC8Repository under examples/FeatureExamples/Basic/PassThrough.
4. Click on the '+' icon next to the PassThrough to show the Streams.
5. (Optional). Click on the '+' icon next to the InTrades stream to view the existing adapter.
6. Open stream viewers for both InTrades and OutTrades by right-clicking on the stream name and then selecting "view stream". Notice the URI in the header of each viewed stream. This URI is used to connect to the stream.
7. Press the "Start Module" button (the green triangle) in Studio's tool bar. After a brief delay, both the InTrades and OutTrades windows start displaying values.
8. Start the input adapter. If the query module is not already running, the streams won't be live and the example input adapter won't be able to connect to them.

Creating an In-process Adapter in C/C++

This section reviews how to write an in-process adapters in C/C++ to communicate with the Sybase CEP Server.

In-process adapters are compiled into dynamically linkable libraries that are called "shared object" libraries on most UNIX-like operating systems. These libraries are linked to the server, and the code in them runs as part of the server process.

This chapter explains how to create simple adapters for reading from and writing to a Sybase CEP Server in the C and C++ programming languages (this API allows you to write both input adapters and output adapters). Since this example is provided in source form, modifications are easily performed.

In-process adapters tend to have lower overhead than out-of-process adapters. They also give you access to some internal features in the engine; for example, you can call a function to find out the schema of a stream.

For an adapter to function, it must be able to attach to its Sybase CEP stream and convert data to (or from) the Sybase CEP Server representation. These two tasks - communication and conversion - are independent of each other. The primary purpose of the In-process Adapter API is to provide the functions needed to communicate with the Sybase CEP Server. You must write your own conversion functions that are appropriate for the source (or destination) of the data.

The Components of an In-process Adapter

A description of the components of an in-process adapter, including the library and ADL files.

This in-process adapter must be linked with Sybase CEP Server and with SDK libraries. Linking is done dynamically at run time.

A complete in-process adapter includes not only a library file that can be linked to the server, but also an associated ADL file; this ADL file specifies the adapter's library file name, the names of the functions in that file, and the adapter properties that may be used with that adapter. Write an ADL file for each in-process adapter and place it in both the Sybase CEP Studio plugins directory and the Sybase CEP Server plugins directory. This is necessary because the Sybase CEP Studio and Sybase CEP Server may execute on different machines across a network.

In addition, place a library containing your in-process adapter code into the bin folder of the server. When the Sybase CEP Server determines that your adapter must be loaded, the Sybase CEP Server code will load your ADL, find the name of the library and entry points in the library, load the library, and begin executing your code.

Algorithm Overview

The C/C++ subfunctions in an in-process adapter run only when called by Sybase CEP Server. This section describes the Sybase CEP Server callback functions.

Each in-process adapter must provide three or four subfunctions that can be called by the server:

- initialize().
- execute().
- shutdown().
- reconnect(), which is only required for a Guaranteed Delivery adapter.

Because these functions must be registered with and called by the server, they are sometimes referred to as callback functions.

The actual names of the functions may be chosen by you; for convenience, we will refer to them as the `initialize()`, `execute()`, and `shutdown()` functions.

The `initialize()` function is called once. This function performs the required initialization steps. For example, if the adapter is an input adapter, the `initialize()` function opens or connects to the data source. If the data source is a file, then the

`initialize()` function will open the input file. The `initialize()` function may also read parameters such as the location of the input and allocate memory.

The `execute()` function is called periodically. If the adapter is an input adapter, the `execute()` function reads one or more rows from the data source, does any required data conversion, and sends that data to a stream. If the adapter is an output adapter, the `execute()` function reads one or more rows from the data stream, does any required data conversion, and sends the data to a data destination.

Note: By calling the `execute()` function periodically, whether there is data or not, the server gives the adapter a chance to check for other conditions, such as timeouts, as well as to check for data.

The `shutdown()` function is called when the server would like to stop the in-process adapter. Typically this is when the server itself is going to shut down. The `shutdown()` function should do any appropriate cleanup, such as closing files, releasing and so on.

When one of these functions is called, the server passes the function a pointer to a C8Adapter object. This C8Adapter object stores information about this particular instance of the adapter. You can have multiple instances of your own adapter running simultaneously, just as you can have multiple copies of a built-in adapter running simultaneously. An adapter pointer is required for many of the calls to the Sybase CEP API. For example, if you want to get the schema of the stream used by a particular instance of an adapter, then you will call the `C8AdapterGetSchema()` function and pass it the C8Adapter pointer that indicates which adapter (and thus which stream) the function should return the schema of.

Each of these functions can call other functions you write as well as various library functions that are part of the server or the In-process Adapter SDK. The library file that contains your in-process adapter contains the three main functions as well as the other functions that support those main functions.

Session State Information and Persistent State Information

This section describes session state and persistent state information, which enables users to use the same information in more than one invocation.

An adapter's `execute()` function can be called many times. Many adapters need to carry over information from one invocation to the next. For example, if you write an output adapter that writes to a file, after you open that output file you will want to store a file pointer to that file so you can write to the file each time without the overhead of reopening the file. We refer to this carried over information as session state, and the Sybase CEP C/C++ SDK provides functions that allow you to store and retrieve session state. Session state information is not carried over if the server crashes or is restarted.

Persistent State information is much like Session State information, except that if Persistence was enabled for the query module that is using this adapter instance, then Persistent State information will persist across system restarts and crashes.

API Interface

This section provides an introduction to the various API interfaces.

API Interface

Sybase CEP provides several header (.h) files. These files help users to declare variables as data types in a platform-independent way, and also contain the function prototypes of the server and SDK library functions that users may call. The header file `c8types.h` provides the data type information.

The API has been divided into the following major sections:

- **Schema:** The Schema API provides meta-information about the data. Information such as the number, names and types of columns are accessible. The schema API is applicable both to in-process adapters and out-of-process adapters, and was described in section *Schema API* on page 118.
- **Message:** Data routing in the Sybase CEP Engine is performed in terms of data packets called messages. A message contains a row of information matching the schema. The API includes functions that give you information about messages. The message API is applicable both to in-process adapters and out-of-process adapters, and was described in section *Message API* on page 121.
- **In-process Adapter:** Every adapter provides an access point for a message to enter or exit the Sybase CEP Engine. Each adapter runs in its own thread. The adapter callbacks (the `initialize()`, `execute()`, and `shutdown()` functions) are always called from this thread.
- **Server:** This contains a few useful functions.

Most in-process adapter code includes:

```
#include "c8types.h"
#include "c8adapter_in_process.h"
#include "c8conversions.h"
#include "c8server.h"
```

Sybase CEP also provides a set of functions that convert between strings and 's internal data types such as TIMESTAMP, FLOAT, and so on.

C/C++ Memory Management API for In-Process and Out-of-Process Operations

The memory management API is the same for in-process and out-of-process operations.

For information on the memory management API, see *Memory Management API* on page 109. Some special cautions apply when these are used with in-process operations; see *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

In-process Adapter API

This section provides an introduction to in-process adapter API signatures, their purposes, and parameters.

These in-process adapter API signatures are defined in `c8adapter_in_process.h`.

Each adapter is executed in its own thread. `initialize()`, `execute()`, and `shutdown()` are always called from this thread.

For input adapters, the will call the `execute()` function function as often as possible.

For output adapters, the Sybase CEP Engine will call the `execute()` function on a periodic basis. Check for message availability; there can be an arbitrary number of messages (0, 1, or more than 1) available for any single call of the `execute()` function. The output adapter is responsible for processing the messages and should then return control to the .

- **const C8Schema *C8AdapterGetSchema(C8Adapter *i_aptr) –** Purpose: Given an adapter pointer, returns the associated schema. The schema specifies the name and data type for each of the columns. The return value is a pointer. Do not change the memory that this pointer points to, `C8Free()` this pointer.

  Parameters:

  - i_aptr: A pointer to the adapter object.

  Returns: A pointer to the schema of this adapter.

The following `GetParam()` functions interface with the ADL files. Please refer to *Adapter Definition Language* on page 319 for detailed descriptions of how to use ADL files. All the GetParam names are displayed in the adapter Properties View in Sybase CEP Studio and are configurable. Parameters can access the original string form as displayed in by calling `C8AdapterGetParamString()`. If you have additional data types `C8AdapterGetParamString()` is the appropriate function call. After obtaining the string, you can call one of the data conversion functions supplied by Sybase CEP.

The name string should be identical to the string provided in the ADL file. For instance, if the ADL file contains

<Parameter Name="TitleRow" xsi:type="xsi:boolean">

then the name should be TitleRow. It is important to match the case. Using an unknown name results in the default value being returned.

Note: In the following functions, the *name parameter can specify a parameter of any data type at all. This can be convenient for user-defined data types. The returned pointer must be released by `C8Free()` to prevent a memory leak.

- **C8Int C8AdapterGetParamInt(C8Adapter *adapter_ptr, const C8Char *name, C8Int default) –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
- name: The name of the adapter property whose value you would like to retrieve.
- default: The default value to return if the adapter property is not set in the project or the ADL file.

Returns: The value of the adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.

- **C8Long C8AdapterGetParamLong(C8Adapter *adapter_ptr, const C8Char *name, C8Long default) –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - name: The name of the adapter property whose value you would like to retrieve.
  - default: The default value to return if the adapter property is not set in the project or the ADL file.

  Returns: The value of the adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.
- **C8Float C8AdapterGetParamFloat(C8Adapter *adapter_ptr, const C8Char *name, C8Float default) –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - name: The name of the adapter property whose value you would like to retrieve.
  - default: The default value to return if the adapter property is not set in the project or the ADL file.

  Returns: The value of the adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.
- **C8Bool C8AdapterGetParamBool(C8Adapter *adapter_ptr, const C8Char *name, C8Bool default); –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
- name: The name of the adapter property whose value you would like to retrieve.
- default: The default value to return if the adapter property is not set in the project or the ADL file.

Returns: The value of the adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.

- **C8Timestamp C8AdapterGetParamTimestamp(C8Adapter *adapter_ptr, const C8Char *name, C8Timestamp default); –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - name: The name of the adapter property whose value you would like to retrieve.
  - default: The default value to return if the adapter property is not set in the project or the ADL file.

  Returns: The value of the adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.
- **C8Interval C8AdapterGetParamInterval(C8Adapter *adapter_ptr, const C8Char *name, C8Interval default); –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - name: The name of the adapter property whose value you would like to retrieve.
  - default: The default value to return if the adapter property is not set in the project or the ADL file.

  Returns: The value of the adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.
- **C8Char *C8AdapterGetParamString(C8Adapter *adapter_ptr, const C8Char *name, C8Char* default); –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

- name: The name of the adapter property whose value you would like to retrieve.
- default: The default value to return if the adapter property is not set in the project or the ADL file.

Returns: A pointer to the value of the specified adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.

- **C8Char *C8AdapterGetParamBlob(C8Adapter *adapter_ptr, const C8Char *name, C8Char* default); –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - name: The name of the adapter property whose value you would like to retrieve.
  - default: The default value to return if the adapter property is not set in the project or the ADL file.

  Returns: A pointer to the value of the specified adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.

  The BLOB values used by this function are raw BLOBs as opposed to hex strings or base64 strings. For an explanation of raw vs. hex string vs. base64 string formats, see *Data Types and Subroutines for UDFs and In-process Adapters* on page 102.
- **C8Char *C8AdapterGetParamXml(C8Adapter *adapter_ptr, const C8Char *name, C8Char* default); –** Purpose: For the specified adapter, reads the value of the named property and returns that value.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - name: The name of the adapter property whose value you would like to retrieve.
  - default: The default value to return if the adapter property is not set in the project or the ADL file.

  Returns: A pointer to the value of the specified adapter property. If there is no adapter property with that name in the project, returns the default value for the property as specified in the ADL file. If the parameter is not defined in either place, returns the value passed as default.
- **C8Bool C8AdapterIsGDSupported(C8Adapter* adapter_ptr); –** Purpose: Determines whether or not this adapter supports Guaranteed Delivery.

  Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

Returns: C8_TRUE if the ADL file for this adapter has the SupportsGuranteedDelivery attribute set to "true", otherwise C8_FALSE.

- **C8Bool C8AdapterConnect(C8Adapter* adapter_ptr, const C8Char* session_id, C8Interval timeout); –** Purpose: Initiates a connection for an adapter that supports Guaranteed Delivery.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - session_id: The ID of the session being established. The ID must be unique.
  - timeout: How long to wait for the connection to be made before timing out.

  Returns: C8_TRUE on success, C8_FALSE if the timeout is reached or an error occurs.
- **const C8Char* C8AdapterGetLastBatchId(C8Adapter* adapter_ptr); –** Purpose: Retrieves the ID of the last batch of messages processed. Used in a Guaranteed Delivery input adapter to determine which messages to send after a reconnect.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: The ID of the last batch processed.
- **void C8AdapterSetLastBatchId(C8Adapter* adapter, const C8Char* i_batch_id); –** Purpose: Sets the ID of the last batch of messages processed. Used in a Guaranteed Delivery output adapter to specify which messages should send after a reconnect.

  Parameters:

  - adapter: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - i_batch_id: The batch ID.

  Returns: Nothing.
- **C8Message *C8AdapterReceiveMessage(C8Adapter *adapter_ptr); –** Purpose: Returns a pointer to a C8Message received by the adapter. If no messages are available, a null pointer is returned.

  When you have finished processing the message, call `C8MessageDestroy(` to release message resources. The user-written `execute()` function should continue to process messages as long as `C8AdapterReceiveMessage()` provides them. When a null is returned from `C8AdapterReceiveMessage()`, the `execute()` function return to Sybase CEP Server.

  This function works only for output adapters.

Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

Returns: A pointer to the message received by the adapter. Returns NULL if there are no new messages.

- **C8Message *C8AdapterReceiveMessageWait(C8Adapter *adapter_ptr, C8Interval interval); –** Purpose: Returns a pointer to a C8Message received by the adapter. If no messages are available, the call will wait for a maximum of "interval" microseconds. If no message is available within interval microseconds, a null pointer is returned. Check the return value to see whether a message arrived or whether a null pointer was returned.

  When you finish processing the message, call `C8MessageDestroy()` to release message resources. The user-written `execute()` function continues to process messages as long as `C8AdapterReceiveMessageWait()` (or `C8AdapterReceiveMessage()`) provides them. When a null is returned from `C8AdapterReceiveMessageWait()`, the `execute()` function returns to Sybase CEP Server.

  This function works only for output adapters.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - interval: The maximum number of microseconds to wait for a message.

  Returns: A pointer to the message received by the adapter. Returns NULL if there are no new messages.

- **C8MessageBatch* C8AdapterGetNextMessagesBatch(C8Adapter* adapter, C8Interval i_timeout); –** Purpose: Retrieves the next batch of messages from . Used in a Guaranteed Delivery output adapter.

  Parameters:

  - adapter: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - i_timeout: How long to wait for the batch of messages before timing out.

  Returns: A pointer to the batch of messages. Returns NULL if there are no new messages

- **void C8AdapterSendMessage(C8Adapter *adapter_ptr, C8Message *msg_ptr); –** Purpose: Sends the specified message to . This function works only for input adapters.

  Parameters:

  - adapter_ptr : A pointer to the C8Adapter object that stores information about this instance of the adapter.

- msg_ptr: The message to send to the server.

Returns: Nothing.

- **void C8AdapterFlushMessages(C8Adapter *adapter_ptr); –** Purpose: Force all queued messages to be sent to the server.

  The `C8AdapterSendMessage()` call does not send messages instantly. Messages are sent at the end of the execute function or during a `C8AdapterSleep()` call. To force the adapter to send all messages that have not yet been sent, call this function.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: Nothing.

- **C8Bool C8AdapterSendMessageListAsBatch(C8Adapter* adapter, C8Message** i_messages, C8SizeType i_num_messages, const C8Char* i_batch_id, C8Interval i_timeout); –** Purpose: Sends the list of messages to Sybase CEP Server as a batch and waits for an acknowledgment that the batch has been processed. This function works only for Guaranteed Delivery input adapters.

  Parameters:

  - adapter: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - i_messages: The array of messages to send to the server.
  - i_num_messages: The number of messages in i_messages.
  - i_batch_id: The ID of this batch of messages.
  - i_timeout: How long to wait for the messages to be processed.

  Returns: C8_TRUE on success; C8_FALSE if the timeout is reached or an error occurs.

  Warning! After you send a particular message pointer, you can not "re-use" that message pointer; in other words, you can not put new values into it and send it again. You must use C8MessageDestroy() to dispose of the message pointer and then use C8MessageCreate() to create a new message pointer when you want to send another message.

  Warning! After you send a msg_ptr, you can not put new values into it and send it again. Use `C8MessageDestroy()` to dispose of the msg_ptr and then use `C8MessageCreate()` to create a new msg_ptr when you want to send another message.

- **const C8UserCredentials* C8AdapterCopyCredentials(C8Adapter* adapter); –** Purpose: Creates a copy of your credentials as set for an adapter. The copy can then be used with other API calls that require credentials. For security reasons, the password is not accessible to the caller.

  Parameters:

- adapter: A pointer to the C8Adapter object.

Returns: A copy of the user credentials, or NULL if the credentials aren't set or an error occurs. You must call `C8UserCredentialsDestroy` to free the memory when you are finished with the credentials.

- **const C8Char * C8AdapterGetName (C8Adapter *adapter_ptr); –** Purpose: Retrieves the ful" adapter name, for example, "workspace/project/submodule1/submodule2/.../adapter".

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: A string that represents the path name of the adapter. This is the CCL or HTTP path name of the instance of the adapter that is running. This is not the pathname of the library file containing the adapter code.
- **C8Float C8AdapterGetPlaybackRate (C8Adapter *adapter_ptr); –** Purpose: Returns the accelerated playback rate. A rate of 1.0 is normal, or non-accelerated.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: The accelerated playback rate. A rate of 1.0 is "normal" (non-accelerated).
- **C8Bool C8AdapterIsPersistent (C8Adapter *adapter_ptr); –** Purpose: Determines whether or not the query module in which this instance of the adapter is running has Persistence turned on.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: C8_TRUE if Persistence is on; C8_FALSE otherwise.
- **C8Bool C8AdapterIsInterrupted (C8Adapter *adapter_ptr); –** Purpose: Determines whether or not the server needs the adapter to return control to the server as soon as possible. Typically, if this function returns C8_TRUE, the adapter should finish processing the current row and return, not continue processing until it runs out of rows.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: C8_TRUE if the server needs the adapter to return control to the server as soon as possible; C8_FALSE otherwise.
- **C8Bool C8AdapterIsConnected (C8Adapter *adapter_ptr); –** Purpose: Determines whether or not the adapter is connected and ready to send (or receive) messages.

Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

Returns: C8_TRUE if the adapter is connected; C8_FALSE otherwise.

- **void C8AdapterLock(C8Adapter *adapter_ptr); –** Purpose: Provides an indivisible sequence of instructions that cannot be interrupted. Users should not perform lengthy operations in a locked state. Nested locks are undefined and should not be used.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: Nothing.
- **void C8AdapterUnlock(C8Adapter *adapter_ptr); –** Purpose: Unlocks a previously locked state. Unlocking an unlocked state is undefined and should not be used.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: Nothing.
- **void C8AdapterSleep(C8Adapter *adapter_ptr, C8Interval us_delay); –** Purpose: Forces the adapter thread to sleeps for the specified amount. This is not normally needed because the Adapter framework provides appropriate synchronization. This function is provided for any special needs.

  The header file c8types.h contains definitions that can be convenient for specifying durations; for example, to delay three seconds, use 3*C8PerSecond. Small intervals are probably inappropriate depending on your CPU speed, system clock resolution, and system loading. In general, on many systems, resolutions less than 100 milliseconds are very imprecise. Even for larger values, the actual length of the sleep is not exactly what was specified.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - us_delay: The number of microseconds to delay.

  Returns: Nothing.
- **void C8AdapterSetSleepInterval(C8Adapter *adapter_ptr, C8Interval us_delay); –** Purpose: Passes the server a parameter that tells the code how long to wait between calls to the `execute()` function. This sleep interval is the time that the CALLER sleeps, not the time that the `execute()` function sleeps. The default is 1 millisecond (1000 microseconds). In most cases, the default value is best; this function is provided for any special needs.

A value of 0 means that the caller will not sleep at all, but will call the `execute()` function again as soon as the previous call to `execute()` has returned.

The header file `c8types.h` contains definitions that can be convenient for specifying durations; for example, to delay three seconds, use 3*C8PerSecond.

As with any sleep-related function, the duration depends on how heavily loaded the system is and the granularity of the system clock. The actual sleep is only an approximation of the amount you request. The actual length of the sleep is likely to be longer than requested if you request a time smaller than the finest granularity of the system clock, or if the system is heavily loaded. The length of the sleep can be less than the amount of time requested if there is a change in server state (for example, a shutdown) or if the adapter is an output adapter and data is available for that output adapter.

Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
- us_delay: The number of microseconds to delay.

Returns: Nothing.

- **C8Interval C8AdapterGetSleepInterval(C8Adapter *adapter_ptr); –** Purpose: This returns a value that indicates how long the server sleeps between calls to the adapter's `execute()` function. This is the time that the CALLER sleeps, not the time that the `execute()` function sleeps. See the description of the `C8AdapterSetSleepInterval()` function for more details.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: A value indicating how long the server sleeps between calls to the adapter's `execute()` function.
- **void C8AdapterSetSessionState(C8Adapter *adapter_ptr, void *session_state) –** Purpose: This stores the session state, which is a set of information that you want to make available across multiple invocations of the adapter's `execute()` function. Although the session state is retained by the Sybase CEP Engine, you are responsible for populating and updating the session state. Storing and retrieving session information is optional. A session state does not contain a size because the session is not persistant over crashes and restarts. A session state is created using `C8Malloc()` or a similar heap-based allocation. Session structures can not be defined on the stack because the stack is popped each time the function exits. For more information, see see comments on `C8AdapterGetSessionState()`. When you are done with the session state, free it. For tips and warnings about allocating and de-allocating memory, see the section titled *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

  In the `shutdown()` function function, the session state should be set to a Null pointer.

Parameters:

- adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
- session_state: A pointer to the "state" information that the user would like to be able to see the next time that the adapter's `execute()` function is called.

Returns: Nothing.

- **void *C8AdapterGetSessionState(C8Adapter *adapter_ptr); –** Purpose: Given a pointer to a C8Adapter object, return the session state information associated with that instance of the adapter. On the initial call to `C8AdapterGetSessionState()`, a null pointer is returned. This indicates no session exists, so you are responsible for creating a session. To construct the session state, call `C8Malloc()` and populate it with your own structure. Using a static or global thread for session states is not safe and the consequences are undefined.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.

  Returns: A pointer to session state information. The exact structure of this information is defined by the you, so the function returns a pointer to the memory without "interpreting" that memory in any way.
- **void *C8AdapterGetPersistentState(C8Adapter *adapter_ptr, C8UInt *o_data_size); –** Purpose: Get a pointer to Persistent State information. if you have information that must be available in subsequent invocations of the adapter, then you can use the `C8AdapterSetPersistentState()` function to tell the server to make that memory persistent, and you can then call this function (`C8AdapterGetPersistentState()`) to retrieve a pointer to that memory. If Studio requests that the stream be persistent, this memory block persists over reboots and system crashes. The o_data_size pointer permits the use of dynamic structure verification providing a convenience utility. Depending on your needs, the o_data_size can be ignored for fixed sized structures. Since persistence depends on the size of your data, calling `C8AdapterRealloc()` might need this information. If you have not already called `C8AdapterSetPersistentState()`, then a call to `C8AdapterGetPersistentState()` returns a null pointer. See also the `C8AdapterSetPersistentState()` function.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - o_data_size: The number of bytes in the user-defined block of memory that was persisted.

Returns: A pointer to the stored information. The exact structure of this information is defined by the you; so thus the function simply returns a pointer to the memory without "interpreting" that memory in any way. Do not free the pointer returned by this function.

- **void C8AdapterSetPersistentState(C8Adapter *adapter_ptr, const void* data, C8UInt data_size); –** Purpose: This allows you to specify that a particular block of memory be made persistent (preserved so that it can be read back on a subsequent invocation of the adapter or even after a crash). The parameter named data must point to the starting address of the block of memory to be preserved. The parameter named "data_size" specifies how many bytes long that block is. The data_size can vary from call to call.

  Only one piece of user memory can be made persistent at a time. If you call this function more than once, then only the pointer passed in the most recent call to `C8AdapterSetPersistentState()` will be returned when you call `C8AdapterGetPersistentState()`. If you need to expand the amount of memory you want to store persistently, use the `C8Realloc()` function to get a new (larger) piece of memory and then call `C8AdapterSetPersistentState()` with the address of that new piece of memory.

  The memory you make persistent is normally memory that you yourself have allocated (for example, via `C8Malloc()`) and therefore you must deallocate via `C8Free()` when you are done with it, If not done beforehand, deallocate persistent memory when the adapter's `shutdown()` function is called. If this memory contains any structures that must be cleaned up (for example, pointers to other memory that must be deallocated, or file handles to files that must be closed, then make sure that you clean those up before you deallocate this memory.

  Parameters:

  - adapter_ptr: A pointer to the C8Adapter object that stores information about this instance of the adapter.
  - data: A pointer to the block of memory containing the bytes to be stored.
  - data_size: The number of bytes of data stored.

  Returns: Nothing.

Below is sample pseudo code showing how `C8AdapterSetPersistentState()` and related functions are typically used.

```
initialize()
{
pMem = C8Malloc(...);
...
C8AdapterSetPersistentState(..., pMem, ...);
...
}
execute()
{
pMem = C8AdapterGetPersistentState(...);
```

```
...
}
```

Remember that if the allocated memory itself contains pointers or handles, those must be cleaned up too.

Server API

An introduction to server API signatures, their purposes, and parameters.

These adapter API signatures are defined in `c8server.h`.

- **C8Char* C8ServerGetPreference(const C8Char *i_section, const C8Char *i_preference, const C8Char *i_default) –** Purpose: When the Sybase CEP Server starts, it reads configuration information from an XML file. By default, this file is named c8-server.conf. You may read the configuration information as of the time that the server started calling `C8ServerGetPreference()`.

  Parameters:

  - i_section: Specifies a particular section of the configuration file, which is an XML file.
  - i_preference: Specifies the name of a particular parameter in that section.
  - i_value: A default that you would like to have returned if the section and preference that you specified are not set in the configuration file.

  Returns: The value stored in the specified preference and section.

  For example, suppose that we want to find the base folder for input and output files used by the adapters supplied by Sybase CEP. The XML for this part of the server configuration file looks like:

  ```
     ...
      <section name="Sybase
  C8/Adapters
  ">
     ...
     <section name="
  ReadWriteBaseFolder
  ">
        <preference name="
  BaseFolder
  " value="/c8test/D5"/>
     </section>
      </section>
  ```

  The value of the section is

  `"SybaseC8/Adapters/ReadWriteBaseFolder"`

  And the value of the preference is

  `"BaseFolder"`

Because one section was nested within another, we composed a single name that included the entire XML path.

Sample C code is below:

```
C8Char i_section[] = "SybaseC8/Adapters/ReadWriteBaseFolder";
    C8Char i_preference[] = "BaseFolder";
    C8Char i_default[] = "";   // Default to empty string.
    C8CharPtr preferenceValue = NULL;
    preferenceValue = C8ServerGetPreference(i_section,
                  i_preference, i_default);
    if (preferenceValue != NULL)  {
   ...
   C8Free(preferenceValue);
   }
```

Note: The value returned by this function is stored in memory allocated by the server. To avoid a memory leak, you must free this memory using the `C8Free()` function.

Session States vs. Persistent States

This section provides more detailed information about session states and persistent states.

A persistent state is a user-defined memory block containing the information necessary to describe connections and perform the adapter function across multiple invocations of the `execute()`execute() function. Connection-related information includes filenames and stream URIs. Additional persistent state information includes file offsets, intermediate calculations that span multiple rows, and processing subtotals or summaries. Persistent state information remains available even after the server crashes. If you call`C8AdapterGetPersistentState()` after a crash, you'll get the same information that you would have gotten before the crash.

A session state is a user-defined memory block containing information that is dependent on transient data items such as file pointers and transient socket connection information. A session state should contain only these transient data items. Session information is not preserved across crashes. If you call `C8GetState()` after a crash, your session state data will not be available.

A pivotal question in distinguishing between session state and persistent state is, "If the system crashed and the in-process adapter was restarted, what would I have to recreate to properly continue my adapter stream?" The persistent state would contain information that does not change while the session state should contain information that must be re-created each time the adapter re-connects to its data sources.

For example, suppose that an adapter is reading from a data file. The adapter starts with the name of the data file, then calls the `fopen()` function to get a file pointer to that file. The adapter also keeps track of the line number of the line that it last read from the file. If the computer crashes and the in-process adapter must resume, the name of the input file remains the same, just like the line number at which to resume reading remains the same, so those two

pieces of information can be stored in the persistent state. The adapter has to re-open the data file and may get a different file pointer than it had previously. The file pointer should therefore be stored in the session state rather than in the persistent state. After a crash, you use the same persistent state information, but mustre-generate session state information.

The following example shows one method of determining when the system has crashed and needs to re-initialize. Notice the suggested sequence at the beginning of the `execute()` function:

```
my_xxx_c8adapter_execute(C8Adapter *i_adapter_ptr)
{
  ...
  // Get the session state
  struct MySessionState *l_session_ptr = (struct MySessionState*)
                       C8AdapterGetSessionState(i_adapter_ptr);
   if( ! l_session_ptr) {
      // The  has been restarted for some reason.
      // Need to recreate the session data.
      if (! my_input_c8adapter_session(i_adapter_ptr)) {
   ...
```

Refer to the *In-Process Input Adapter* on page 178and *In-process Output Adapter* on page 181 sections for more details.

Suggested Session and Persistent State Initialization

This section describes session and persistent state initialization in the event of a system crash.

The session state is set up in the initialization routine. If the Sybase CEP Engine or the system crashes, the session state will return a null pointer when `C8AdapterGetSessionState()` is called. You must then perform whatever initialization is necessary to properly process information. In the initialization call, files may need to be opened, permissions tested, environments probed and tested, and so on. Similarly, when the Sybase CEP Engine restarts, the session state is again missing and the same initialization sequence must be performed. The only difference between an initial Sybase CEP Engine start and a restart is that the persistent state should be present on a restart, an initial start has no persistent state.

For non-trivial adapters, a session state and persistent state may resemble the following skeleton code:

```
// Perform session initialization
C8Bool my_xxx_c8adapter_session(C8Adapter *i_adapter_ptr)
{
  // The user must define this struct!
  struct MySessionState *l_session_ptr =
        (struct MySessionState*)C8Malloc(
            (C8UInt) sizeof(struct MySessionState));
  // Open files, sockets, db connections, ...
  // Specific information in the session
  l_session_ptr->m_file = fopen(...);
```

```
  ...
  C8AdapterSetSessionState(i_adapter_ptr, (void*)l_session_ptr);
  return C8_TRUE;
}
// Perform adapter initialization
C8Bool my_xxx_c8adapter_initialize(C8Adapter *i_adapter_ptr)
{
  struct MyPersistentData *l_persistent_ptr =
        (struct MyPersistentData *)C8Malloc(
            (C8UInt) sizeof(struct MyPersistentData));
  // Test ptrs for existence - possibly error out
  ...
  // Populate persistent state - Read parameters, test file
  // existence, permissions, etc
  ...
  // Create session state. Open files and record session
  if ( ! my_xxx_c8adapter_session(i_adapter_ptr)) {
        // Could not initialize session!
        C8ErrorSet( ... );
        return C8_FALSE;
  }
  // Save persistent state
  C8AdapterSetPersistentState(i_adapter_ptr,
(void*)l_persistent_ptr,
        sizeof(struct MyPersistentData));
  return C8_TRUE;
}
```

The initialization call provides a proper session state, while the `execute()` function uses the session state as well as the persistent state. The first call to `C8AdapterGetPersistentState()` in the initialization function will return a null pointer. You must create whatever memory structure is necessary to perform proper adapter functions. From this point on, `C8AdapterGetPersistentState()` will return a pointer to the last saved persistent state.

Signatures of User Functions

Each user-defined adapter must provide three or four entry points in order to function. Introduces these adapter functions, their purposes, and parameters.

The entry points are identical for both input and output adapters, though the internal logic will differ. Refer to the *In-Process Input Adapter* on page 178and *In-process Output Adapter* on page 181 sections and the examples for details on function implementation.

While the actual function names are arbitrary, the names must match the names specified in the ADL file. If the names are missing or do not match, a fatal runtime error will result.

For example, part of the ADL file may look like:

```
<LibraryName>
my_input_adapter_lib
</LibraryName>
  <InitializeFunction>
```

```
my_input_adapter_init
</InitializeFunction>
  <ExecuteFunction>
my_input_adapter_execute
</ExecuteFunction>
  <ShutdownFunction>
my_input_adapter_shutdown
</ShutdownFunction>
  <ReconnectFunction>
my_input_adapter_reconnect
</ReconnectFunction>
```

The corresponding C function signatures would be:

```
C8Bool my_input_adapter_init(C8Adapter* i_adapter_ptr);
C8Bool my_input_adapter_execute(C8Adapter* i_adapter_ptr);
void my_input_adapter_shutdown(C8Adapter* i_adapter_ptr);
C8Bool my_input_adapter_reconnect(C8Adapter* i_adapter_ptr);
```

- **C8Bool my_input_c8adapter_init(C8Adapter *i_adapter_ptr) –** Purpose: This initialization function prepares the adapter for operation.

  You can perform file opening and socket connections in anticipation of adapter execution. Depending on adapter needs, the session state may or may not be created.

  Parameters:

  - i_adapter_ptr: A pointer to a set of information about this instance of the adapter.

  Returns: If the adapter is properly initialized, the routine will return C8_TRUE. If the adapter cannot initialize, you can issue appropriate error messages, release any allocated Sybase CEP resources, and return C8_FALSE. If the Sybase CEP Engine receives a C8_FALSE, the adapter isconsidered inactive and no further communications will be attempted.
- **C8Bool my_input_c8adapter_execute(C8Adapter *i_adapter_ptr) –** Purpose: The Sybase CEP Engine will periodically call the user-defined adapter for execution. Write the internals of the `execute()` function according to your needs. The `execute()` function is called repeatedly.

  Parameters:

  - i_adapter_ptr: A pointer to a set of information about this instance of the adapter.

  Returns: On successful processing, the execute function will return C8_TRUE. As long as the execute function returns C8_TRUE, the execute function will be called again. If a problem is encountered, you are expected to issue appropriate error messages and return C8_FALSE. The Sybase CEP Engine will then call the shutdown function, call the initialization function, and call the execute function exactly one more time. If this last call to the execute function returns C8_FALSE, the adapter is considered dead and will no longer be called. This logic exists to accommodate cases of disconnection.
- **void my_input_c8adapter_shutdown(C8Adapter *i_adapter_ptr); –** Purpose: The Sybase CEP Engine determines when the adapter must be shut down. Either the associated

stream is being shut down by the engine or the execute function needs reinitializing. Perform whatever actions are necessary to close the adapter; this may involve closing files, sockets, and database connections. If the Sybase CEP Engine is closing the stream, the thread of the adapter is terminated and no further calls will be made to these user functions.

Parameters:

- i_adapter_ptr: A pointer to a set of information about this instance of the adapter.

Returns: Nothing.

- **C8Bool my_input_c8adapter_reconnect(C8Adapter *i_adapter_ptr); –** Purpose: The Sybase CEP Engine determines that the adapter is attempting to perform a Guaranteed Delivery function but is currently disconnected. Your adapter should reconnect with Sybase CEP Server with `C8AdapterConnect`. For an input adapter, request the last batch ID. For an output adapter, specify the last batch ID. This function is only used with Guaranteed Delivery.

  Parameters:

  - i_adapter_ptr: A pointer to a set of information about this instance of the adapter.

  Returns: C8_TRUE on successful reconnection, C8_FALSE on failure.

  Warning! The adapter SDK does not restrict the number of times the reconnect function is called, and it does not sleep between reconnects. In order to prevent a loop, you should take precautions to avoid repeatedly attempting to reconnect and creating a loop.

In-process Input Adapters

This section describes the intialization, execution, and shutdown subroutines for in-process input adapter, and provides an outline of the user code for an input adapter.

An in-process input adapter gets data from a source and sends the data to the Sybase CEP Engine. The sample code performs the outlined operations as well as other demonstration routines. The purpose of the sample code is to create a random input adapter that will work for any schema. The sample code also provides various routines to access parameters of various data types and print schemas among other things. These functions demonstrate the various API calls.

As we described, an in-process input adapter must have at least three subroutines:

Initialization: This may includes activities such as opening sockets, verifying the existence of files and/or directories, and pinging network connections. Both persistent and session data should be created. To make it easier to handle restarts, we recommend that the code for creating and storing session data be put in a separate subroutine.

Execution: This involves a query for any input to send to the Sybase CEP Engine. If input is available, messages get created and sent to the input stream. Whether one or multiple messages are sent per invocation depends on expected behavior of the adapter. If no input is

available, the routine should call `C8AdapterSleep()` and specify a short time period, for example, one second. Alternatively, the function can return and wait until it is called again.

Shutdown: This occurs when the Sybase CEP Engine receives information that the adapter should shut down. This may be from someone clicking the "Stop" button in the Studio, or from another source such as an end of a data file. The shutdown routine should perform appropriate shutdown operations pertinent to the adapter: closing sockets, closing database connections, closing files, and so on. Release any resources obtained from the Sybase CEP Engine in order to complete the shutdown. In particular, call C8AdapterSetSessionState(adapter_ptr, NULL) to allow the Sybase CEP Engine to release internal resources.

An outline of the user code for an input adapter:

```
// Perform session initialization
C8Bool my_input_c8adapter_session(C8Adapter *i_adapter_ptr)
{
  struct MySessionState *l_session_ptr =
        (struct MySessionState*)C8Malloc(
         (C8UInt) sizeof(struct MySessionState));
  // Open files, sockets, db connections, ...
  // Specific information in the session
  l_session_ptr->m_file = fopen(...);
  ...
  C8AdapterSetSessionState(i_adapter_ptr, (void*)l_session_ptr);
  return C8_TRUE;
}
C8_Bool my_input_c8adapter_initialize(C8Adapter *i_adapter_ptr)
{
  struct MyPersistentData *l_persistent_ptr =
        (struct MyPersistentData*)C8Malloc(
         (C8UInt) sizeof(struct MyPersistentData));
  // Perform session initialization
  if ( ! my_input_c8adapter_session(i_adapter_ptr) ) {
      C8ErrorSet(MY_ERR_CODE, "Cannot initialize ...");
      return C8_FALSE;
  }

  // Perform persistent data initialization
  if ( ! l_persistent_ptr ) {
      C8ErrorSet(MY_ERROR_CODE, "Cannot obtain memory...");
      return C8_FALSE;
  }
  // Get user parameters from Studio. Notice these may be placed
  // in persistent data or re-read as required.
  l_persistent_ptr->m_ms_delay =
    C8AdapterGetParamInt(i_adapter_ptr, (const C8Char*)"MsDelay",
              (C8Int)10);
  ... other parameters setup ...
  // Save the persistent state
  C8AdapterSetPersistentState(i_adapter_ptr, l_persistent_ptr,
      sizeof(struct MyPersistentData));
  return C8_TRUE;
}
```

```
C8_BOOL my_input_c8adapter_execute(C8Adapter * i_adapter_ptr)
{
   // Get the session state
   struct MySessionState *l_session_ptr = (struct MySessionState*)
                       C8AdapterGetSessionState(i_adapter_ptr);
   if ( ! l_session_ptr) {
      // The Sybase CEP Engine has been restarted for some reason.
      // Need to recreate the session data.
      if (! my_input_c8adapter_session(i_adapter_ptr)) {
          // Cannot create session!
          C8ErrorSet(...);
          return C8_FALSE;
      }
     l_session_ptr = (struct MySessionState*)
                       C8AdapterGetSessionState(i_adapter_ptr);
     if ( ! l_session_ptr ) {
          // Cannot retrieve newly created session state!
          C8ErrorSet(...);
          return C8_FALSE;
      }
   }
   // Get the persistent data for operations.
   ...
   // Get data from the data source and publish the data to Sybase CEP
   ...
   return C8_TRUE;
}
// Close any input files, socket connections, database
// connections, etc.
void my_input_c8adapter_shutdown(C8Adapter *i_adapter_ptr)
{
    C8UInt l_data_size = 0;
    MySessionState *l_session_ptr = (struct MySessionState*)
        C8AdapterGetSessionState(i_adapter_ptr);
    MyPersistentData *l_persistent_ptr =

(MyPersistentData*)C8AdapterGetPersistentState(i_adapter_ptr,
         &l_data_size);
    if(l_persistent_ptr) {
        printf("Closing data source. Lines processed: %d\n",
            l_persistent_ptr->m_number_lines_read);
       C8AdapterSetPersistentState(i_adapter_ptr, (void *)NULL, 0);
    } else {
        printf("Cannot get persistent data in shutdown!\n");
    }
    C8AdapterSetSessionState(i_adapter_ptr, NULL);
    if(l_session_ptr) {
        C8Free((void *)l_session_ptr);
    } else {
        printf("Cannot get session data in shutdown!\n");
    }
    return;
}
```

In-process Output Adapters

This section describes the intialization, execution, and shutdown callbacks for in-process output adapters, and provides an example of a typical callback pattern for output adapters.

A typical in-process output adapter gets data from the Sybase CEP Engine and publishes to some destination. The tutorial output adapter, like the tutorial input adapter, provides additional routines of interest for adapter developers.

The three callbacks for an output in-process adapter are:

Initialization: Initialize the adapter in anticipation of receiving messages from the Sybase CEP Engine and sending the messages to some external data sink. This includes such activities as opening sockets, verifying the existence of files and/or directories, and pinging network connections. Any relevant state information created must be saved.

Execution: The user queries for any messages. If a message is available, execution callback extracts relevant information from the message and handles the information according to your adapter code. This includes, but is not limited to, creating CSV and XML files, filtering according to some criteria, and redirecting to some data sink. The exact mechanism is highly dependent on your intentions. Whether one or multiple messages are processed depends on the Sybase CEP message queue. If no messages are available, the routine will usually return to the caller. Note that the output adapter execute function may be called even if no messages are available. You may be interested in maintaining connections, querying external events, and so on.

Shutdown: The Sybase CEP Engine has receives information that the adapter should shut down either from someone clicking the "Stop" button in Studio, or from another source. The shutdown routine should perform whatever shutdown operations pertain to the adapter: closing sockets, closing database connections, closing files, flushing buffers, and so on. Release any resources obtained from the Sybase CEP Engine. In particular, call C8AdapterSetSessionState(adapter_ptr, NULL) to allow the Sybase CEP Engine to release internal resources.

A typical pattern for an output adapter is:

```
C8Bool my_output_c8adapter_initialize(C8Adapter *adapter_ptr)
{
  // Do initialization if needed. This includes session and
  // persistent data creation. See the input adapter for examples.
}
C8Bool my_output_c8adapter_execute(C8Adapter *adapter_ptr)
{
   C8Message *msg = NULL;
   // The output adapter setup is the same as the input adapter.
   ...
   // Receive messages and process.
   // As long as the engine provides messages, the adapter
   // should process them.
```

```
   while (C8AdapterIsInterrupted(adapter_ptr) != C8_TRUE   &&
          C8AdapterReceiveMessage(adapter_ptr) != NULL) {
      // Publish message to the external user destination
      ...
      // Destroy message
      C8MessageDestroy(msg);
   }
   return C8_TRUE;
}
void my_output_c8adapter_shutdown(C8Adapter *adapter_ptr)
{
  // The shutdown matches the pattern in the input adapter.
}
```

In-process Adapters

This section provides more information about the initialize(), execute(), and shutdown() routines required in a C/C++ in-process adapter.

Initializating in-process adapters in C/C++

The tutorial ADL file on initialization enables you to work with each Sybase CEP data type.

Each of these values may be entered in a Properties View (in Sybase CEP Studio) and retrieved with the sample code:

```
DumpParamName(i_adapter_ptr, (C8Char*)"Filename");
DumpParamInt(i_adapter_ptr, (C8Char*)"UserDefinedInteger");
...
```

The parameter name gets associated with the value from the adapter's Properties View in Sybase CEP Studio. This printing of parameter values may be observed by displaying an instance of the adapter in Sybase CEP Studio. Make sure Sybase CEP Server is properly connected, then press "Start Module". The adapter initialization routine is called and the parameter printing routines display the Sybase CEP Studio values on Sybase CEP window.

To change the values and have Sybase CEP Server notice the changes, press "Stop Module" in the Sybase CEP Studio window. Change the value of the parameters and press "Start Module" when you are finished. The Sybase CEP Server log will contain the values that were entered into the adapter's Properties View in Sybase CEP Studio. This exercise demonstrates proper communication between the Sybase CEP Studio and the Sybase CEP Server.

Another useful printout for initialization is the current working directory of the server. Because Sybase CEP Studio can execute on a different machine than Sybase CEP Server, it is sometimes confusing to determine the working directory of Sybase CEP Server. Printing the directory helps with this problem.

A session state may be required if you need to initialize variables, open file/sockets/db connects, and so on. Define a single block of memory to store this data. This means that you cannot use such structures as hash maps or linked lists without formatting the data structures into and out of a single memory block.)

The initialization returns C8_TRUE to indicate a proper initialization sequence. If the code is not able to properly initialize the adapter, C8_FALSE is returned. Reasons for improper initialization include input files not found, permissions problems and connection problems.

A fatal initialization sequence should provide details to Sybase CEP Server log file in a manner that is adequate to resolve the issue.

Returning C8_TRUE continues the adapter execution while C8_FALSE produces an error condition visible on Sybase CEP Studio.

Important: The call to the `initialize()` function is a blocking call. Some server operations may not run while the `initialize()` function is running, and therefore you should write your `initialize()` function so it finishes quickly (preferably in less than one second).

Execution function of input adapters

This section describes the execution function for input adapters, which retrieves data from external sources, creates, and processes messages.

For an in-process input adapter, Sybase CEP Server will repeatedly call the execute function. The period between calls cannot be reliably predicted since it depends heavily on system load. Each adapter runs in its own thread and should limit the amount of time processing input data. An input adapter should not block on waiting for input.

The execution function of input adapters is to retrieve data from an external source, create a Sybase CEP message, and send that message to the Sybase CEP Engine. An input adapter may send more than one message per invocation, but should not send an excessive number of messages. As a general rule, sending more than 100 messages per execute call is likely not a good idea. If the number of messages per invocation of the execute module is important, add a "number of messages" parameter to the ADL so that Studio will allow easy tuning of your application.

Message creation is an important part of execution. Empty messages are created as:

```
// Wait a limited time for data to show up in the data source
if (MyDataSourceSelect(i_adapter_ptr, l_ds, l_ds->m_ms_delay)) {
    /* Create a message to send to the Sybase CEP Engine */
    l_message_ptr = C8MessageCreate(C8_MESSAGE_POSITIVE,
                                    i_schema_ptr);
    if(l_message_ptr == 0) {
        printf("ERROR: Cannot create positive message!!\n");
        return C8_FALSE;
    }
    // Arbitrarily create a Null record every 50th line.
    if( (l_ds->m_number_lines_read % 50) == 0) {
        l_is_null = C8_TRUE;
    }
    MyCreateRandomMessage(l_ds, l_message_ptr, i_schema_ptr,
        l_is_null);
    C8AdapterSendMessage(i_adapter_ptr, l_message_ptr);
```

```
    ++l_ds->m_number_lines_read;
    C8MessageDestroy(l_message_ptr);
}
```

The function `MyDataSourceSelect()` returns true when data is available. A message must be created to send to the Sybase CEP Engine; `C8MessageCreate()` performs the task of constructing an empty message. The message gets populated with `MyCreateRandomMessage()` that fills each column according to datatype with random data, then the message gets sent by `C8AdapterSendMessage()`. To illustrate setting message fields to null, every fiftieth message will null all the fields. This can be observed in the input Studio stream viewer.

After the Sybase CEP Engine receives the message, the message is processed. Your adapter must destroy the message as seen by the adapter using `C8MessageDestroy()`. Failure to perform this function will result in a memory leak.

Because the persistent state has changed, it must be saved. The function `C8AdapterSetPersistentState()` saves the state.

The execution function finishes by returning C8_TRUE. This indicates to Sybase CEP Server that all is well and no fatal conditions have occurred. Problems that might arise include socket timeouts, file permissions, broken connections, and anything that is not recoverable. These fatal conditions return C8_FALSE.

If the execution function returns C8_FALSE, the following sequence is performed once in an attempt to recover:

1. The shutdown function is called. This should properly close/terminate the adapter.
2. The initialization function is called. This is an attempt to re-initialize the adapter.
3. The execute function is called once again. If this call fails, no further attempts at revival are performed.

Shutdown in C/C++

This section describes the shutdown function and other activities that you should perform as part of shutdown.

The shutdown function prepares to release all resources the adapter has acquired. Actions such as buffer flushing, file closing, socket and/or database disconnection should be performed. You might wish to print an activity summary on Sybase CEP Server log.

The shutdown is similar to:

```
void user_input_c8adapter_shutdown(C8Adapter *i_adapter_ptr)
{     C8UInt l_data_size = 0;
    MySessionState *l_session_ptr = (struct MySessionState*)
        C8AdapterGetSessionState(i_adapter_ptr);
    MyPersistentData *l_persistent_ptr =

(MyPersistentData*)C8AdapterGetPersistentState(i_adapter_ptr,
           &l_data_size);
```

```
    if(l_persistent_ptr) {
        printf("Closing data source. Lines processed: %d\n",
            l_persistent_ptr->m_number_lines_read);
        C8AdapterSetPersistentState(i_adapter_ptr, (void *) NULL,
0);
    } else {
        printf("Cannot get persistent data in shutdown!\n");
    }
    // Close the session state
    C8AdapterSetSessionState(i_adapter_ptr, NULL);
    if(l_session_ptr) {
        C8Free((void*)l_session_ptr);
    } else {
        printf("Cannot get session data in shutdown!\n");
    }
    return;
}
```

Session state and persistent state are retrieved from the Sybase CEP Engine. The number of lines processed is displayed on the Sybase CEP window.

The `C8AdapterSetSessionState()` call releases the Sybase CEP Engine resources that were acquired for this adapter.

Important: The call to the `shutdown()` function is a blocking call. Some server operations may not run while the `shutdown()` function is running, and therefore you should write your `shutdown()` function so that it finishes quickly (preferably in less than one second).

Useful Utility Functions

This section contains utility functions that you may find useful.

Printing the Schema

Describes the function in Sybase CEP for printing schema and its syntax.

Sending or receiving data with the Sybase CEP Engine requires a schema for the message containing the data. Sybase CEP Studio creates a schema interactively with the user. When the user interfaces to this data, the schema provides information such as column names and column data types.

Try to develop adapters that are flexible enough to accommodate change, rather than hard-coding a schema. A major first step in this flexibility is obtaining the existing schema. The following function simply prints the existing schema. This function is identical for both an input and an output adapter:

```
static
void PrintSchema(C8Adapter* i_adapter_ptr)
{
    const C8Schema* i_schema_ptr =
C8AdapterGetSchema(i_adapter_ptr);
    C8Char *l_name;
```

```
    const char *l_type_name;
    C8UInt ndx;
    C8UInt l_col_cnt;
    if(i_schema_ptr == 0) {
        printf("Cannot get schema pointer.\n");
        return;
    }
    l_col_cnt = C8SchemaGetColumns(i_schema_ptr);
    printf("This schema has %d columns.\n", l_col_cnt);
    for (ndx = 0; ndx < l_col_cnt; ++ndx) {
        l_name = (C8Char*)C8SchemaGetColumnName(i_schema_ptr, ndx);
        l_type_name = MapC8TypeToString(C8SchemaGetColumnType(
                                            i_schema_ptr, ndx));
        printf("\t%s\t%s\n", l_name, l_type_name);
    }
}
```

This routine takes the adapter pointer and retrieves the schema pointer. The `C8SchemaGetColumns()` function provides the number of columns in a schema and provides the upper limit to the print loop. Column names are etrieved with `C8SchemaGetColumnName()` and the Sybase CEP datatypes with another utility function `C8SchemaGetColumnType()`. The output of this function will be displayed on the same window as Sybase CEP Server since adapters run tightly bound to Sybase CEP Server.

A minor utility function, `MapC8TypeToString()` converts a Sybase CEP data type to a displayable string.

```
// Given a C8_TYPES object, return a printable string.
static
const char *MapC8TypeToString(C8_TYPES i_atype)
{
    switch(i_atype) {
        case C8_INT:       return "C8_INT";
        case C8_LONG:      return "C8_LONG";
        case C8_FLOAT:     return "C8_FLOAT";
        case C8_CHAR_PTR:  return "C8_CHAR_PTR";
        case C8_TIMESTAMP: return "C8_TIMESTAMP";
        case C8_INTERVAL:  return "C8_INTERVAL";
        case C8_BOOL:      return "C8_BOOL";
        case C8_XML:       return "C8_XML";
        case C8_INVALID:   // fall through to default
        default:           return "C8_INVALID";
    }
    // Should not happen
    return "C8_INVALID";
}
```

Printing a Parameter

Sybase CEP Studio provides a Properties View for each instance of an adapter. Each instance can have different run-time parameters. This section provides an example for printing parameters.

There can be multiple instances of each adapter present. Each instance of your in-process adapter is run in a separate thread.

Access to the individual values for each parameter ensures proper configuration. This parameter value retrieval is normally performed as an initialization step. Each parameter must be retrieved according to its own datatype. The specific datatype retrieved corresponds to the xsi:name in the ADL file. The correspondence is shown in the table below:

C8_TYPES	**xsi:name**
C8_CHAR_PTR	xsi:string
C8_INT	xsi:integer
C8_LONG	xsi:long
C8_BOOL	xsi:boolean
C8_TIMESTAMP	xsi:timestamp
C8_INTERVAL	xsi:interval
C8_FLOAT	xsi:double

While there is currently no API function to retrieve the data type of a parameter in the ADL file, parameter data types change infrequently. Attempting to retrieve a parameter with the incorrect data type results in an error message on Sybase CEP Server log and an undefined value returned to the user SDK.

An example of printing C8Int data types would be:

```
static
void DumpParamInt(C8Adapter*i_adapter_ptr, C8Char*i_param_name)
{
    C8Int l_param_value;
    l_param_value = C8AdapterGetParamInt(i_adapter_ptr,
                    i_param_name, 0);
    printf("Parameter Int %s=%d\n", i_param_name, l_param_value);
}
```

In the `C8AdapterGetParamInt()` call, the third parameter is the default parameter. If the SDK cannot find the parameter for some reason, this value becomes the function return value. Default parameters exist because users can choose not to enter these values on the adapter's Properties View in Sybase CEP Studio. It is not currently possible to determine if the value has been defaulted or if the user has chosen not to assign a value to the parameter.

Obtaining data from other data types is involves the same process with different `GetParam()` data type function calls.

Requirements for the C/C++ File

This section describes function and syntac requirements for the C/C++ file.

Your source code will need to #include the proper headers.

```
// Include the Sybase CEP header files.
#include "c8types.h"         // Definitions of C8Float, C8Bool, etc.
#include "c8adapter_in_process.h"
#include "c8conversions.h"    // Access to funcs like
FloatToString()
#include "c8server.h"
```

Your in-process adapter must contain `initialize()`, `execute()`, `shutdown()`, and, if your adapter supports Guaranteed Delivery, `reconnect()` functions. You can choose any names for these functions as long as those names don't duplicate the names of other functions, but you must tell the compiler and linker to make those names visible externally and to use the C naming conventions instead of the C++ naming conventions. To do this, your C file should have the following near the beginning:

```
// Ensure that functions are "exported" properly from dll.
#if defined(_MSC_VER)
#define USER_ADAPTER_EXPORT __declspec( dllexport )
#else // defined(_MSC_VER)
#define USER_ADAPTER_EXPORT
#endif // defined(_MSC_VER)
// forward declarations of callback functions for
// the in-process adapter
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
USER_ADAPTER_EXPORT
C8Bool user_input_adapter_initialize(C8Adapter *adapter_ptr);
USER_ADAPTER_EXPORT
C8Bool user_input_adapter_execute(C8Adapter *adapter_ptr);
USER_ADAPTER_EXPORT
void   user_input_adapter_shutdown(C8Adapter *adapter_ptr);
// Reconnect callback required only if your adapter
// supports Guaranteed Delivery
USER_ADAPTER_EXPORT
C8Bool user_input_adapter_reconnect(C8Adapter *adapter_ptr);
```

In addition, your in-process adapter should contain the following at the end:

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

The combination of these two pieces of code tells the compiler that the functions you define between the "extern C {" and the closing "}" should be externally visible and should use the C naming convention, not the C++ name-mangling convention.

Use one of the examples that Sybase CEP ships as a starting point. Find the examples under the SybaseC8Repository at `examples/FeatureExamples/Adapters/InProcess`.

Step-by-Step Instructions for Creating an In-process Adapter

This section describes the steps required to create an in-process adapter, from writing a C or C++ program to copying the ADL file to the Sybase CEP Server and Sybase CEP Studio plugins directories.

1. Write the C or C++ program.
2. Create a project file or a script file to specify what to compile and link. This is explained in more detail below.
3. Compile and link the program to a dynamically linkable library such as .dll in Microsoft Windows or .so in UNIX-like operating systems.
4. Create the .ADL file that describes the adapter's parameters, and that specifies the name of the library file (.dll or .so) that you created and the names of the entry points (the initialize, execute, shutdown, and, for Guaranteed Delivery, reconnect routines) in that library file.
5. Copy the library file to the Server's bin directory.

   On Microsoft Windows, this is typically:

   ```
   C:\Program Files\SybaseC8\server\bin
   ```

   On UNIX-like operating systems, this is typically:

   ```
   /home/<userid>/sybasec8/server/bin
   ```
6. Copy the .adl file to the server's plugins directory.
7. Copy the .adl file to studio's plugins directory.

   On Microsoft Windows, this is typically:

   ```
   C:\Program Files\SybaseC8\Studio\plugins
   ```

   On UNIX-like operating systems, this is typically:

   ```
   /home/<userid>/SybaseC8/Studio/plugins
   ```
8. Stop and restart both Studio and the Server.

Compiling an In-process Adapter

This section describes the steps required and settings you need to modify in order to compile an in-process adapter.

To compile your in-process adapter, you must specify appropriate settings for your compiler, including:

1. Specify that the compiler should generate a shared object file (.so) or a dynamic linkable library (.DLL) file.
2. Your list of "include" directories should contain the directory that holds `c8adapter.h`.
3. The list of libraries that you link to should include `c8_sdk_server_lib.lib` on Microsoft Windows or `c8_sdk_server_lib.so` on UNIX-like operating systems.

   If you are on Microsoft Windows, you can use Microsoft's Visual Studio. If you are using Visual Studio, perform the following actions to compile your in-process adapter:
4. Start Microsoft Visual Studio.
5. Create a project file by going to the menu and clicking on File -> New -> Project.
   a) Click on the [+] to expand "Visual C++ Projects".
   b) Click on "Win32".
   c) In the right-hand pane, Click on "Win32 Project".
   d) Fill in the name that you want to use for your project.
   e) Browse and specify the directory in which you want the project to be stored.
   f) Click "OK".
   g) When the "Win32 Application Wizard" window appears,click on "Application Settings", then click on "DLL".
   h) Click on "Finish".
6. Microsoft Visual Studio creates a simple .cpp file to use as a starting point. We recommend removing all contents of this file and inserting your own C code for the in-process adapter. Make sure that your code contains "#includes" and forward declarations described in the section titled *Requirements for the C/C++ File* on page 188.
7. If you have other C-language source files that you need, add them to the project
8. Update several settings that are available in the "Property Pages" for this project.
   a) Update the list of directories to search for include files.

      To do this, go to the menu, click on "Project" and then on "MySample Properties".

      You will get a new window titled "MySample Property Pages". In the left-hand pane of this window, click on "Configuration Properties", then on "", and then on "General".

      The right-hand pane should now show a list of settings that you can modify. Click in the field to the right of "Additional Include Directories" and add the directory that contains the c8adapters.h file which is included with the Sybase CEP product.

On Microsoft Windows, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk\c\include
```

On64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\SybaseC8\Server\sdk\c\include
```

You can add other directories if necessary for your in-process adapter.

b) Turn off precompiled headers. To do this, go to the left-hand pane in the "Property Pages" window, click on "" and then on "Precompiled headers", then click on "Create/Use Precompiled Header" and set it to "Not Using Precompiled Headers".

c) Add the Sybase CEP library files. To do this, go to the left-hand pane of the "Property Pages" window, click on "Linker" and then on "General".

   In the field to the right of "Additional Library Directories", add the directory that contains the library.

   On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk\c\lib
```

   On Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\SybaseC8\Server\sdk\c\lib
```

d) Tell the linker not to include debugging information. To do this, go to the left-hand pane of the "Property Pages" window, click on "Linker" and then on "Debugging". For the field "Generate Debugging Info", change the value to "No".

e) Add a dependency on the `c8_sdk_server_lib.lib` file. To do this, go to the left-hand pane of the "Property Pages" window, click on "Linker" and then on "Input".

   In the field to the right of "Additional Dependencies", enter

```
c8_sdk_server_lib.lib
```

   (Note that you do not need to enter the complete path; entering the file name is sufficient.)

   If you are using Microsoft's Visual C development and environment and you'd like to double check that you haven't skipped a step, you can look at the "Command Line" for the C/C++ compiler and the "Command Line" for the Linker.

   To view the command line for the compiler, go to the left-hand pane of the Property Pages window, click on "C/C++" and then click on "Command Line". The command line should look similar to the following:

```
/Od /I "C:\Program Files\SybaseC8\Server\sdk\c\include"
  /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_USRDLL"
  /D "INPROCESSOUTPUTADAPTER4_EXPORTS" /D "_WINDLL"
```

```
  /D "_UNICODE" /D "UNICODE" /Gm /EHsc /RTC1 /MDd
  /Fo"Debug\\" /Fd"Debug\vc80.pdb" /W4 /nologo /c
  /Wp64 /ZI /TP /errorReport:prompt
```

If you set the warning level to a value other than 3, then the "/W3" will be different.

The command line can or can not include

```
/D "_DEBUG"
```

If the command line includes this, you can only be able to use the .DLL on a computer that has the debug version of the C runtime library. (For more information, see the *Troubleshooting* on page 351 section.)

To view the command line for the linker, go to the left-hand pane of the Property Pages window, click on "Linker" and then click on "Command Line". The command line should look like the following:

```
/OUT:"C:\c8test\E2\C_SDK\Adapter4\Adapter4\Debug
\Adapter4.dll"
  /INCREMENTAL /NOLOGO /LIBPATH:"C:\Program Files
\SybaseC8\Server\sdk\c\lib"
  /DLL /MANIFEST /MANIFESTFILE:"Debug
\Adapter4.dll.intermediate.manifest"
  /SUBSYSTEM:WINDOWS /MACHINE:X86 /ERRORREPORT:PROMPT
c8_sdk_server_lib.lib
  kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib
  shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib
odbccp32.lib
```

Now that you have entered all the project properties, click the "OK" button on the "Property Pages" window. At this point you should be ready to compile.

9. To compile, use the appropriate option on the "Build" menu (for example, "Build MySample").
10. Copy the DLL file to the server's "bin" directory.
11. If you are using a command-line compiler (such as "cc"), please perform the following steps:
    a) Create and execute a script file with commands similar to the following. The example below is based on the "cc" compiler found on many UNIX-like operating systems:

```
cc  -I${HOME}/sybasec8/server/sdk/c/include  \
    -o libmma.so  \
    -L${HOME}/sybasec8/server/sdk/c/lib/  \
    -lc8_sdk_server_lib  \
    -fPIC  \
    -shared  \
    mma.c
```

where "mma.c" is the name of your source code file and "libmma.so" is the name you want to use for the library file.

- I specifies the directory(s) to be searched for "#include" files.
- o specifies the name of the output file or the shard library file.
- L specifies the directory(s) to search for library files that need to be linked with this one.
- l specifies the name of the Sybase CEP library to link with in order to use Sybase CEP functions such as `C8GetFloat()`
- shared specifies that the output will be a shard object lirary file as opposed to, for example, a stand-alone executable program.

Warning! Although on UNIX-like operating systems your library file name is typically of the form "libXYZ.so" (for example, "libmma.so"), your .adl file should specify only "mma" as the library name. Do not specify "libmma.so" in the .adl file. The .adl file "interpreter" makes platform-specific adjustments for the filename extension (.so vs. .dll) and, if necessary, an initial "lib" prefix.

a) Copy the compiled shared library file to the server's "bin" directory.
Copy the .adl file to both the server's plugins directory and the Studio's plugins directory.

12. Copy the compiled shared library file to the server's "bin" directory.

13. Copy the .adl file to both the server's plugins directory and Studio's plugins directory.

Alternatively, use a makefile or other technique to compile the file.

Practical Tips for Using the In-process Adapter SDK

This section provides tips for debugging adapters and improving the performance of adapters.

Testing and Debugging

This section provides tips useful for testing and debugging your in-process adapter.

In the initialization code, query and print the parameters that Sybase CEP Studio provides through the `C8AdapterGetParamInt()` type calls. This provides assurance of proper communication and updating with Sybase CEP Studio.

The initialization function is also a good place to print the current working directory. The sample code provides an #ifdef to call the proper system function switch regardless of whether you are on Microsoft Windows or one of the UNIX-like operating systems.

Since your in-process adapter is running as part of the server, if your code contains assertions, you will stop the server as well as your adapter. You can use asserts during the development and debugging states but should avoid using them in production.

Note: While some of the example code provided by Sybase CEP contains assertions, you should remove them or replace them with other error-handling code before you move your system from development to production.

You can create and use an ADL file without writing the C/C++ code for the in-process adapter. To do this, put the ADL file in Studio's plugins directory and use it in Sybase CEP Studio. This

allows you to do some basic testing of the ADL file and also to show the parameters to potential users and allow them to comment.

Sybase CEP Server and Sybase CEP Studio only read ADL files and library files when they start, so you must restart both the Server and Sybase CEP Studio any time you copy an updated ADL file to the plugins directory or copy an updated library file to the bin directory.

All parameters known to Sybase CEP Studio for a specific ADL instance can be printed to stdout by a call to `PrintParamsAsStrings()`. This can be useful in resolving some kinds of problems. A discrepancy between expected and display values may indicate a missing update for an adapter or library file. Experimenting with exceptionally large or small values can also encounter overflow/underflow problems.

If there are problems with obtaining typed parameters from Sybase CEP Studio, parameters of any data type can be retrieved as a string with `C8AdapterGetParamString()`. This could be useful for user data types that do not conform to Sybase CEP data types. You can display the data type as a string, retrieve with `C8AdapterGetParamString()` and convert with a specialized conversion routine. Remember to `C8Free()` the returned pointer.

If you use the Persistence feature, the Server stores persistent data in a directory named storage. This is why a stream will begin immediately upon server invocation even when Sybase CEP Studio has not selected the "Start Stream" function. This is a feature and not a bug. To stop this action for debugging, remove the directory name storage in the server execution directory. Do this only on a development system, not a production system. These directories may be different for release and debug versions. Another way is to select the "Run with clean slate" from the Sybase CEP Studio "Run" pull-down menu. This will cause persistence to be reinitialized before execution begins.

In the course of development, the storage directory should be cleared. If the stream is persisted and not removed, schema changes or normal starting and stopping can display erratic behavior. Removing the storage directory requires re-creation of the workspace as well.

In the development cycle, adapters have to be modified.

1. Because Sybase CEP Studio and Sybase CEP Server cache adapter information, stop both.
2. Copy the adapter's ADL file to the adapter directories.
3. Restart Sybase CEP Studio.
4. Go to the display of the adapter and remove, then reattach, the adapter to the stream.

If display information was modified, the adapter should now reflect the update. If there was an ADL coding error such as misspelled XML elements, Sybase CEP Studio will refuse to load the adapter and an error message will appear at Sybase CEP Studio startup.

`C8AdapterGetParamString()` can be used to differentiate between a zero value and any empty string. If the parameter field is an integer, and `C8AdapterGetParamInt()` is called, there is no difference between a 0 value and an empty field. By calling `C8AdapterGetParamString()` you can detect the difference.

To avoid generating overwhelming amounts of data during early testing phases, consider calling the `C8Sleep()` function between each row you send. When testing an output adapter, use an input adapter that is configured to send data at a slow rate.

The libraries distributed by Sybase CEP do not contain debugging information but a debugger can be used.

Performance Optimizations

This section contains tips for optimizing the performance of your adapter.

- In most cases, in-process adapters process multiple rows (if available) each time the `execute()` function is called. This can dramatically improve performance. The Server runs each adapter on a separate thread. Each thread runs in a loop that does the following:
  1. Call the adapter's `execute()` function, and waits for the `execute()` function to return.
  2. Sleep. By default, on most platforms the thread sleeps for 1 millisecond (1/1000 of a second).

  This means that the `execute()` function can be called no more than 1000 times per second. If the adapter processes only one row per call, then the adapter can process no more than 1000 rows per second.

  If data is available to be processed, the adapter should try to process multiple rows per call. This can increase performance by a factor of tens or hundreds. However, the adapter should not process so many rows that it prevents the server from having enough CPU time, either.

  Experiment to find the optimal number of rows to process per call, depending on your hardware, the other processes that are running on your computer. A general rule is to try to process 30-100 rows per call and then experiment with increasing or decreasing the number of rows per call to find where performance is best.

  You can also write the adapter so that rather than processing a fixed number of rows per call, the adapter will listen for interrupt requests from the server. Your adapter can call the function `C8AdapterIsInterrupted()` to see whether the server would like the adapter to yield control of the CPU. See *In-process Adapter API* on page 161 for a description of the function. See *In-process Output Adapters* on page 181 for sample code that uses this function.
- When you want to change the number of rows sent per call to the `execute()` function. In some cases, you may also want to adjust the sleep interval in between calls to the `execute()` function. This sleep value is used by the framework between calls to the execute function. You can get and set this sleep interval value by using the following functions:
  - C8AdapterGetSleepInterval().
  - C8AdapterSetSleepInterval().

  These functions are defined in `c8adapter_in_process.h`.

Multi-Stream In-Process Adapters

Although most adapters connect to a single stream, it is possible to connect an adapter to multiple streams.

This may be useful if, for example, the processing load for each row is relatively heavy, and you want to spread the processing workload across multiple Sybase CEP Servers (more precisely, across multiple containers working under the same manager). To do this, you can use Sybase CEP's distributed (parallel) query feature and write a single adapter that will write to multiple input streams.

It is also possible to write a single adapter that will read from multiple output streams, which allows you to merge data from multiple streams. In most cases the easiest way to merge streams is through CCL rather than writing a custom adapter.

To write to multiple streams, you can use an out-of-process adapter. This is relatively straightforward once you know how to read from (or write to) a single stream. However, to maximize performance, you may wish to have a single in-process adapter write to read from (or write to) multiple streams. For an example of an in-process adapter that writes to multiple streams, look at the `c8_multistream_adapter.cpp` example, which you can find in:

```
C:\Program Files\SybaseC8\Server\sdk\c\examples
\c8_multistream_adapter.cpp
```

or

```
/home/<userid>/sybasec8/server/sdk/c/examples/
c8_multistream_adapter.cpp
```

This example uses C++ features and therefore cannot be compiled as a pure C program.

The term "managed" adapter in the source code refers to an "in-process" adapter.

Although the adapter uses a combination of in-process and out-of-process API calls, the adapter is considered an in-process adapter and generally must follow the rules for an in-process adapter, including:

- The list of #include files should not include client (out-of-process) files such as `c8client.h.`
- You should not link with client libraries, such as `c8_sdk_client_lib.`
- You should call the `C8ClientSDKInitialize()` and `C8ClientSDKShutdown()` functions appropriate for in-process adapters, and should not call the versions that are appropriate for output adapters.

Troubleshooting complier and linker error messages

This section describes how to avoid compiler and linker error messages.

If you see a compiler/linker error message similar to the following:

```
error ... second C linkage of overloaded function
'C8ClientSDKInitialize' not allowed
```

you are probably trying to #include both the server-side #include and the client-side #include. To avoid this problem, make sure that both your #includes and your list of libraries for the linker exclude client libraries.

Setting Up Dynamic Queries and Streams with the C/C++ SDK

The Sybase CEP C/C++ SDK includes a way to register a query.

Note: Before reading this section, please read *Creating Streams and CCL Statements from Inside A Program* on page 24 if you have not already done so.

Registering a query includes:

- Creating the streams specified in the registered query.
- Creating the statements specified in the registered query.
- Compiling the CCL statements.
- Loading the query into the server.
- Connecting ("binding") the stream names in the CCL statements to the URI's of existing streams, or creating new streams.
- Starting execution of the query.

To register a query, use the `C8RegisterQuery()` function.

You must use a C8StreamInfo object to specify information about each of the streams used in the query. The information about each stream must include the following:

- The name of the stream as used in the query.
- Whether the stream is used as output stream, input stream, or local stream.
- The CCL URI of the stream (for input or output streams only).

Important: The project whose streams are bound by the registered query binds must be up and running for the `C8RegisterQuery()` function to succeed.

If databases or user-defined extensions are used in the queries, the databases and extensions must be configured in the server and in the compiler for the query to work properly.

The query can use any of the streams in a project (query module) as well as any streams the query defines locally. The only restriction is that the local streams must be defined in such a

way that the CCL compiler can deduce their schemas automatically. Also, the streams defined locally for this query are not available in any other queries. The query can also create new input streams and output streams.

Note: LOCAL streams and LOCAL windows are not listed in the results of query tools.

The API

An introduction to SDK functions, their purposes, and parameters.

The functions available in the C/C++ SDK to register a query are listed below:

- **C8Status C8RegisterQuery (const C8Char * i_server_uri, const C8Char *i_workspace, const C8Char *i_query_name, const C8Char *i_query_text, C8SizeType i_stream_cnt, const C8StreamInfo ** i_streams, C8SizeType i_parameter_cnt, const C8ParameterInfo **i_parameters, const C8CompilerOptions *i_opts, C8Bool i_cleanup_tmp_files, C8Char *** o_tmp_file_list); –** Purpose: Register a query.

  Parameters:

  - i_server_uri: The URL of the (for example, http://localhost:6789) that manages the query.
  - i_workspace: The name of the workspace where the query is run.
  - i_query_name: The name of the query. This name must be unique among the dynamically loaded queries for this workspace.
  - i_query_text: The text of the query. The text of the query is a set of zero or more CCL statements. Any CCL constructs may be used in the query.
  - i_stream_cnt: The number of streams in the query.
  - i_streams: An array of objects describing the streams used by the query.
  - i_parameter_cnt: The number of parameters in the query.
  - i_parameters: An array describing the query parameters. If you are not specifying any parameters (for example, "$XYZ" in a CCL statement), you can pass NULL.
  - i_opts: A struct describing the compiler options to be set. Pass NULL to use defaults.
  - i_cleanup_tmp_files: A boolean indicating whether to clean up temporary files.
  - o_tmp_file_list: A pointer that will be set to point to an array of character strings that contain the names of the temporary files. If you do not want the names of these files, simply pass NULL.

    Note: this is a pointer to a pointer to a pointer (3 asterisks) since it's a pointer to an array of arrays of characters. If a non-null pointer to a C8Char ** is passed, the location is filled with a newly allocated, NULL-terminated array of NULL-terminated strings. Note that the array may be filled (and therefore will need to be freed) even if the function does not return success.

  Returns: C8_OK on success; C8_FAIL otherwise.

- **C8StreamInfo * C8StreamInfoCreateInputBound(const C8Char * i_name, const C8Char * i_src_uri, C8Interval i_max_delay, C8Bool i_is_out_of_order, C8Interval i_out_of_order_delay, C8Bool i_is_use_server_timestamp); –** Purpose: Creates an instance of C8StreamInfo describing a bound input stream.

  Parameters:

  - i_name: The stream name as used in the CCL query text.
  - i_src_uri: The URI of the stream this stream binds to.
  - i_max_delay: Max delay parameter for the input stream, in microseconds.
  - i_is_out_of_order: C8_TRUE if the stream is to accept out-of-order messages; C8_FALSE otherwise.
  - i_out_of_order_delay: If out-of-order messages are accepted, the maximum out-of-order delay, in microseconds.
  - i_is_use_server_timestamp: C8_TRUE if the stream is to set the timestamp of incoming messages to the current server time; C8_FALSE otherwise.

  Returns: An instance of the C8StreamInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8StreamInfoDestroy()`.
- **C8StreamInfo * C8StreamInfoCreateInputUnbound(const C8Char * i_name, const C8Schema * i_schema, C8Interval i_max_delay, C8Bool i_is_out_of_order, C8Interval i_out_of_order_delay, C8Bool i_is_use_server_timestamp); –** Purpose: Creates an instance of C8StreamInfo describing an unbound input stream.

  Parameters:

  - i_name stream: The name as used in the CCL query text.
  - i_schema: The stream schema.
  - i_max_delay: Max delay parameter for the input stream, in microseconds.
  - i_is_out_of_order: C8_TRUE if the stream is to accept out-of-order messages; C8_FALSE otherwise.
  - i_out_of_order_delay: If out-of-order messages are accepted, the maximum out-of-order delay, in microseconds.
  - i_is_use_server_timestamp: C8_TRUE if the stream is to set the timestamp of incoming messages to the current server time.

  Returns: An instance of C8StreamInfo if successful; NULL otherwise
- **C8StreamInfo * C8StreamInfoCreateOutputUnbound(const C8Char * i_name, const C8Schema * i_schema); –** Purpose: Creates an instance of C8StreamInfo describing an unbound output stream.

  Parameters:

  - i_name: The stream name as used in the CCL query text.
  - i_schema: The stream schema.

Returns: An instance of the C8StreamInfo if successful; NULL otherwise. The object returned by this function must be deallocated with C8StreamInfoDestroy().

- **C8StreamInfo * C8StreamInfoCreateOutputBound(const C8Char * i_name, const C8Char * i_dst_uri); –** Purpose: Creates an instance of C8StreamInfo describing a bound output stream.

  Parameters:

  - i_name: The stream name as used in the CCL query text.
  - i_dst_uri: The URI of the stream this stream binds to.

  Returns: An instance of the C8StreamInfo if successful; NULL otherwise. The object returned by this function must be deallocated with C8StreamInfoDestroy().
- **C8StreamInfo * C8StreamInfoCreateLocal(const C8Char * i_name); –** Purpose: Creates an instance of C8StreamInfo describing a local stream.

  Parameters:

  - i_name: The stream name as used in the CCL query text.

  Returns: An instance of the C8StreamInfo if successful; NULL otherwise. The object returned by this function must be deallocated with C8StreamInfoDestroy().
- **void C8StreamInfoDestroy(C8StreamInfo * i_stream_info); –** Purpose: Deallocates a C8StreamInfo object.

  Parameters:

  - i_stream_info: The object to destroy.
- **C8ParameterInfo * C8ParameterInfoCreateInteger (const C8Char * i_name, C8Int i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing an integer query parameter.

  Parameters:

  - i_name: The parameter name.
  - i_value: The parameter value.

  Returns: An instance of the C8ParameterInfo if successful; NULL otherwise.
- **C8ParameterInfo * C8ParameterInfoCreateFloat (const C8Char * i_name, C8Float i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing a query parameter of type float.

  Parameters:

  - i_name: The parameter name.
  - i_value: The parameter value.

  Returns: An instance of the C8ParameterInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8ParameterInfoDestroy()`.

- **C8ParameterInfo * C8ParameterInfoCreateString (const C8Char * i_name, const C8Char * i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing a query parameter of type string.

  Parameters:

  - i_name: The parameter name.
  - i_value: The parameter value.

  Returns: Instance of the C8ParameterInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8ParameterInfoDestroy()`.
- **C8ParameterInfo * C8ParameterInfoCreateBoolean (const C8Char * i_name, C8Bool i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing a query parameter of type boolean.

  Parameters:

  - i_name: The parameter name.
  - i_value: The parameter value.

  Returns: An instance of the C8ParameterInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8ParameterInfoDestroy()`.
- **C8ParameterInfo * C8ParameterInfoCreateLong (const C8Char * i_name, C8Long i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing a query parameter of type long.

  Parameters:

  - i_name: The parameter name.
  - i_value: The parameter value.

  Returns: An instance of the C8ParameterInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8ParameterInfoDestroy()`.
- **C8ParameterInfo * C8ParameterInfoCreateTimestamp (const C8Char * i_name, C8Timestamp i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing a query parameter of type timestamp.

  Parameters:

  - i_name: The parameter name.
  - i_value: The parameter value.

  Returns: An instance of the C8ParameterInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8ParameterInfoDestroy()`.
- **C8ParameterInfo * C8ParameterInfoCreateInterval (const C8Char * i_name, C8Interval i_value); –** Purpose: Creates an instance of a C8ParameterInfo describing a query parameter of the type timestamp.

  Parameters:

- i_name: The parameter name.
- i_value: The parameter value.

Returns: An instance of the C8ParameterInfo if successful; NULL otherwise. The object returned by this function must be deallocated with `C8ParameterInfoDestroy()`.

- **void C8ParameterInfoDestroy (C8ParameterInfo * i_prm_info); –** Purpose: Deallocates a C8ParameterInfo object.

  Parameters:

  - i_prm_info: The C8ParameterInfo object to destroy.

Registering a query: Example

This section contains an example program that registers a query.

The outline of the program is:

1. Initialize the SDK. This step is required before calling other functions in the SDK.
2. Compile a project named "streamkeeper", which contains the streams that our dynamic query binds.
3. Start executing the streamkeeper project.
4. Create StreamInfo objects that contain the information required to bind the registered query's streams to the streamkeeper project's streams.
5. Create a SybaseC8ParameterInfo object to store information about the parameter ($VolumeThreshold) that is referenced in the registered query's CCL statement(s).
6. Register the query.
7. Publish some messages/rows.
8. Read those messages/rows.
9. Stop the registered query.
10. Stop the project whose streams you are bound to.
11. Clean up.

The example code below is a modified subset of the example provided with the SDK.

```
/**
 *  Example for registering a query in C
 * Copyright (C) 2006, Sybase, Inc. All rights reserved.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "nspr.h" /* for PR_snprintf */
#include "c8adapter.h"
#include "c8compiler.h"
#include "c8status.h"
#include "c8reggry.h"
```

```
int
example_register_query_main(int argc, char **argv)
{
    /*
     * This is an example of using the Sybase CEP C API
     * for registering queries.
     */
    /* Before this example can be run, Sybase CEP Server
     * must be running, and a workspace named "Default"
     * must exist on it (create the workspace with
     * Sybase CEP Studio or the c8_client application).
     */
    /* declare necessary variables */
    int ret = -1;
    const C8Char * workspace_name = "Default";
    // This is the name of a 'project' that will have
    // streams that our registered query can bind to.
    const C8Char * stream_keeper_name = "StreamKeeper";
    const C8Char * in_stream_name = "InTrades";
    const C8Char * out_stream_name = "OutTrades";
    const C8Char * server_hostname = "localhost";
    const C8Char * server_port = "6789";
    C8Char server_uri [1024];
    C8Char ccl_uri_of_input_stream [1024];
    C8Char ccl_uri_of_output_stream [1024];
    const C8Char * query_name = 0;
    const C8Char * query_text = 0;
    // For each stream that the registered query uses,
    // we must provide a description in the form of a
    // C8StreamInfo object.  In this example, our
    // registered query will use 3 streams: an Input
    // stream, an Output stream, and a local stream.
    C8StreamInfo *streams[3] = { 0,0,0 };
    // In this example, we will use a CCL parameter
    // named $VolumeThreshold, e.g. in the CCL statement:
    //    INSERT INTO ...
    //    SELECT ...
    //    FROM
    //    WHERE Volume > $VolumeThreshold;
    // We will define a SybaseC8ParameterInfo object that
    // contains information about that $VolumeThreshold
    // parameter.
    C8ParameterInfo *parameters[1] = {0};
    C8Subscriber * sub = 0;
    C8Publisher * pub = 0;
    C8Schema * schema = 0;
    // The message/row that we will send.
    C8Message * msg_1 = 0;
    // The message/row that we expect to receive.
    C8Message * rcv_msg_1 = 0;
    // This will point to an array of strings that
    // contain the names of the temporary files that
    // will be used during the compile-and-register
    // process.
    C8Char ** tmp_files = 0;
    C8Timestamp now = 0;
```

```
    const C8Char * str = 0;
    /* Initialize the SDK; must do this exactly once per
     * process.  Note that NSPR and other libraries are
     * initialized within this call as well.
     */
    if (C8_OK != C8ClientSDKInitialize(argv[0], 0)) {
        return -1;
    }
    // Compose the URI of the server.
    PR_snprintf(server_uri, 1024, "http://%s:%s",
        server_hostname,
        server_port);
    /* First, we need a module that defines our streams.
     * The StreamKeeper.ccl is an example of such a
     * module.For the purposes of this example, we will
     * compile and start it here; however, in a typical
     * application this will be done elsewhere.
     */
    if (C8_OK != C8Compile("stream-keeper.ccl", "stream-
keeper.ccx",
        workspace_name, "StreamKeeper", 0)) {
        fprintf(stderr, "%s: Could not compile StreamKeeper\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    if (C8_OK != C8StartProgram(server_uri, workspace_name,
            "stream-keeper.ccx")) {
        fprintf (stderr, "%s: Could not start StreamKeeper\n",
                 argv[0]);
        ret = -1;
        goto cleanup;
    }
    C8Sleep(5*C8PerSecond); /* give it some time to settle */
    /* Determine the URIs for the StreamKeeper's streams.
     * Typically, these would be hard-coded or passed as
     * parameters.
     */
    PR_snprintf(ccl_uri_of_input_stream, 1024,
        "ccl://%s:%s/Stream/%s/%s/%s",
        server_hostname, server_port,
        workspace_name, stream_keeper_name, in_stream_name );
    PR_snprintf(ccl_uri_of_output_stream, 1024,
        "ccl://%s:%s/Stream/%s/%s/%s",
        server_hostname, server_port,
        workspace_name, stream_keeper_name, out_stream_name );
    /*
     * Now, that the environment is set up, on to the core
     * of the example!
     *
     * The StreamKeeper contains two streams, InTrades and
     * OutTrades, but no queries. The query we will be
     * registering in this example filters the data on its
     * input stream and forward the query results into its
     * output stream.
     *
```

```
     * The query uses three streams:
     *      MyStrIn (input),
     *      MyStrOut (output), and
     *     MyStrLocal (a local stream for the query's own use).
     * We need to define the array of StreamInfo
     * in order to bind these streams appropriately.
     */
    streams [0] = C8StreamInfoCreateInputBound(
                    "MyStrIn", ccl_uri_of_input_stream,
                    2*C8PerSecond, C8_FALSE, 0, C8_FALSE);
    streams [1] = C8StreamInfoCreateLocal("MyStrLocal");
    streams [2] = C8StreamInfoCreateOutputBound(
                    "MyStrOut", ccl_uri_of_output_stream);
    if (0 == streams[0] ||
        0 == streams[1] ||
        0 == streams[2]) {
        fprintf (stderr,
                 "%s: Could not create stream info objects\n",
                 argv[0]);
        ret = -1;
        goto cleanup;
    }
   parameters[0] = C8ParameterInfoCreateFloat("VolumeThreshold",
            100.0);
    if (0 == parameters[0]) {
        fprintf (stderr,
                "%s: Could not create parameter info objects\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    /*
     * Note, that the registerQuery() call determines the schema
     * for the input and output streams automatically by querying
     * the server (that's why the module containing the streams,
     * in our case StreamKeeper, must be running).
     * There can be more than one input stream and more than
     * one output stream defined for the query.
     *
     * A note about "local" streams. There can be 0 or more local
     * streams defined for the query (depending on the query).
     * The query does not define the schema for the MyStrLocal
     * explicitly: the query must be written in such a way that
     * the compiler can determine the local streams' schemas
     * automatically. (See Sybase CEP Programmer's Guide
         and
                    Sybase CEP CCL Reference Guide
        for details on how to write CCL in such a way that
     * the compiler can deduce the schemas for the local streams.)
     */
    query_name = "MyFilter";
    query_text =
                "INSERT INTO "
                "    MyStrLocal "
                "SELECT * "
                "FROM "
```

```
                "     MyStrIn "
                "WHERE "
                "     Volume > $VolumeThreshold;\n"
                "INSERT INTO "
                "     MyStrOut "
                "SELECT * "
                "FROM MyStrLocal;\n"
                ;
    if (C8_OK != C8RegisterQuery(server_uri, workspace_name,
        query_name, query_text, (C8SizeType) 3,
        (const C8StreamInfo **) streams, (C8SizeType) 1,
        (const C8ParameterInfo **) parameters,
        0, C8_FALSE, & tmp_files)) {
        fprintf (stderr, "%s: Could not register query\n",
                 argv[0]);
        ret = -1;
        goto cleanup;
    }
    if (tmp_files) {
        /* tmp_files contains null-terminated array of strings;
         * let's print it out! later, we'll have to free all the
         * strings as well as tmp_files.
         */
        C8Char ** pp = tmp_files;
        for (; *pp; ++pp) {
           fprintf(stderr, "Created temporary file: %s\n", *pp);
        }
    }
    /* The official example is now over! Well, almost over (we
     * still need to be able to stop the module we just started).
     * However, let's make sure it works.  There are no
     * adapters connected to the StreamKeeper module, so we
     * must (a) generate some data, and (b) receive the output.
     */
    C8Sleep(5*C8PerSecond); /* let things settle a bit */
    /* Create a subscription first (note, that the stream URI
     * is the URI from the StreamKeeper module).
     */
    sub = C8SubscriberCreate(ccl_uri_of_output_stream);
    if (0 == sub) {
        fprintf (stderr, "%s: Could not subscribe to stream\n",
                 argv[0]);
        ret = -1;
        goto cleanup;
    }
    /* Create a publisher (note, that the stream URI is the
     * URI from the StreamKeeper module).
     */
    pub = C8PublisherCreate(ccl_uri_of_input_stream);
    if (0 == pub) {
        fprintf (stderr,
                 "%s: Could not create a publisher to stream\n",
                 argv[0]);
        ret = -1;
        goto cleanup;
    }
```

```
    C8Sleep(3*C8PerSecond); /* wait for things to settle */
    schema = C8GetStreamSchema(ccl_uri_of_input_stream);
    if (0 == schema) {
        fprintf (stderr, "%s: Could not determine stream schema
\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    /* Now, publish some data */
    msg_1 = C8MessageCreate(C8_MESSAGE_POSITIVE, schema);
    if (0 == msg_1) {
        fprintf (stderr, "%s: Could not create message\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    now = C8Now();
    if (C8_OK !=
         C8MessageColumnSetStringByName(msg_1,"Symbol","IBM") ||
        C8_OK !=
         C8MessageColumnSetFloatByName(msg_1,"Price",50.11) ||
        C8_OK !=
         C8MessageColumnSetIntByName(msg_1,"Volume",120) ) {
        fprintf (stderr, "%s: Could not set message data\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    C8MessageSetMessageTimestamp(msg_1, now);
    if (C8_OK != C8PublisherSendMessage(pub, msg_1)) {
        fprintf (stderr, "%s: Could not publish message\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    /* let's now receive the message */
    rcv_msg_1 = C8SubscriberGetNextMessage(sub, 20*C8PerSecond);
    if (0 == rcv_msg_1) {
        fprintf (stderr, "%s: didn't receive first message\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    /* check that the expected messages came through: */
    if (C8_OK != C8MessageColumnGetStringByName(rcv_msg_1,
            "Symbol", &str)) {
        fprintf (stderr, "%s: could not read message value\n",
                argv[0]);
        ret = -1;
        goto cleanup;
    }
    if (strcmp (str, "IBM")) {
        fprintf (stderr, "%s: unexpected message value received
\n",
                argv[0]);
```

```
        ret = -1;
        goto cleanup;
    }
    /* disconnect publisher and subscriber */
    C8SubscriberDestroy(sub);
    sub = 0;
    C8PublisherDestroy(pub);
    pub = 0;
    /* unregister query by stopping it */
    if (C8_OK != C8StopProgram(server_uri, workspace_name,
            query_name)) {
        fprintf (stderr, "%s: could not stop program %s\n",
                argv[0], query_name);
        ret = -1;
        goto cleanup;
    }
    /* stop the master query */
    if (C8_OK != C8StopProgram(server_uri, workspace_name,
            "StreamKeeper")) {
        fprintf (stderr, "%s: could not stop program %s\n",
                argv[0], "StreamKeeper");
        ret = -1;
        goto cleanup;
    }
    ret = 0; /* completed successfully! */
cleanup:
    if (ret != 0) {
        C8SizeType l_errtxtlen = C8ErrorGetMessageLength();
        C8Char * l_errbuf = 0;
        if (l_errtxtlen > 0) {
            l_errbuf = (C8Char *) C8Malloc(l_errtxtlen);
            *l_errbuf = 0;
            C8ErrorGetMessageText(l_errbuf, l_errtxtlen);
            fprintf(stderr, "Error message: %s\n", l_errbuf);
            C8Free(l_errbuf);
            l_errbuf = 0;
        }
    }
    /* release all resources */
    if (sub) { C8SubscriberDestroy(sub); sub = 0; }
    if (pub) { C8PublisherDestroy(pub); pub = 0; }
    if (0 != ret) {
        C8StopProgram(server_uri, workspace_name, query_name);
        C8StopProgram(server_uri, workspace_name,
"StreamKeeper");
    }
    if (streams[0]) {
        C8StreamInfoDestroy(streams[0]); streams[0] = 0;
        }
    if (streams[1]) {
        C8StreamInfoDestroy(streams[1]); streams[1] = 0;
        }
    if (streams[2]) {
        C8StreamInfoDestroy(streams[2]); streams[2] = 0;
        }
    if (parameters[0]) {
```

```
        C8ParameterInfoDestroy(parameters[0]); parameters[0] = 0;
        }
    if (msg_1) {
        C8MessageDestroy(msg_1); msg_1 = 0;
        }
    if (rcv_msg_1) {
        C8MessageDestroy(rcv_msg_1); rcv_msg_1 = 0;
        }
    if (schema) {
        C8SchemaDestroy(schema); schema = 0;
        }
    if (tmp_files) {
        C8Char ** pp = tmp_files;
        for (; *pp; ++pp) {
            C8Free(*pp);
            *pp = 0;
        }
        C8Free(tmp_files);
        tmp_files = 0;
    }
    /* and shut down the library */
    if (C8ClientSDKShutdown() != C8_OK) {
        /* could not shutdown the library */
        ret = -1;
    }
    return ret;
}
```

The complete source code is in the `example_register_query.c` program.

On Microsoft Windows, if you installed to the default directory, the file(s) will be in:

`C:\Program Files\sybasec8\server\sdk\c\examples`

On UNIX-like operating systems, if you installed to the default directory, the file(s) will be in:

`/home/<userid>/sybasec8/server/sdk/c/examples`

The example uses two additional files, named stream-keeper.ccl and stock-trades.ccs, which are in the same directory.

Instructions for compiling and linking this code are very similar to the instructions for compiling and linking an out-of-process adapter. Because this example uses some code in the nspr library, you must also perform the following actions:

- Add "`.../sybasec8/server/sdk/c/include/nspr`" or "`...\sybasec8\server\sdk\c\include\nspr`", depending upon your operating system, to the list of include directories ("..." represents the directory in which you installed Sybase CEP).
- Add `nspr4.lib` and `plc4.lib` to the list of library files. For example, in MS Visual Studio, add this to the "Additional Dependencies" in the "Linker/Input" section.

Creating Streams Dynamically

In earlier sections, we showed how to bind the query to existing streams. You can also create your own streams. The approach is similar to that shown in the Java SDK chapter.

Troubleshooting

Provides examples of errors you may encounter with C/C++.

Sample error cases:

- **The project that contains the streams you wish to bind to is not running**
  The program is unable to find information about the streams in the project (including tuple descriptors) and load the program into the workspace. The exception will most likely indicate a failed SOAP call. Check that the server is actually running, the server URL you are passing is the correct URL for the server (it must include the correct hostname and port number), and that all the project has been started successfully.
- **Compiler Errors**
  These errors result from invalid CCL files or CCL Syntax errors. The easiest way to troubleshoot this kind of error is by compiling the same CCL files with .

Control: Compile/Start/Stop/Status

This section describes how to compile, start, and stop a project.

Important: C8ClientSDKInitialize() must be called prior to calling any of these functions.

Compiling a CCL Project

Write a C/C++ program that calls the Sybase CEP compiler to compile a Sybase CEP project.

A Sybase CEP project is a `.ccp` file and the files that it references. If you have just a .ccl file without a `.ccp` file, you can also compile that. You can also compile a schema file (a `.ccs` file).

The C/C++ SDK supports only compilation of projects and `.ccl` query modules.

When you use the compiler API, you can set the same options (for example, turning on debug mode, turning off warnings) as you can set when you use the standalone c8_compiler program.

The Compiler API

The `c8compiler.h` file contains function prototypes for the Sybase CEP Compiler API, a set of functions that enable you to compile a Sybase CEP project file (.ccp) or schema file.

The primary functions in this API are the `C8Compile()` function and the `C8RemoteCompile()`C8RemoteCompile() function, which perform the same actions as the Sybase CEP compiler (c8_compiler on UNIX or c8_compiler.exe on Microsoft

Windows). They translate a group of CCL statements into a form the server can use. You can specify the same options for `C8Compile()` as you can specify on the command line of the Sybase CEP compiler. (For a description of the command-line parameters of the c8_compiler program, see *Compile a Project or a Schema File* on page 71.)

The API provides additional functions that you may find useful if you want to specify options for the compiler, such as turning warning messages on or off. For a full description of these options, see *Compile a Project or a Schema File* on page 71, and especially the first table in that section.

Important: Prior to calling any function from the client SDK, the SDK initialization function `C8ClientSDKInitialize()` must be called. See the descriptions of the `C8ClientSDKInitialize()` and `C8ClientSDKShutdown()` functions for more information.

The functions in the compiler API can be called from any out-of-process program. These functions may not be called from inside an in-process task, such as an in-process adapter or an in-process UDF.

The functionality provided by this compiler API is similar in the following Sybase CEP SDKs:

- C/C++
- Java 1.5
- .NET3

The Sybase CEP CCL compiler API is shown below.

- **C8Status C8Compile(const C8Char * i_input_file, const C8Char * i_output_file, const C8Char * i_workspace, const C8Char * i_load_name, const C8CompilerOptions * i_opts); –** Purpose: Compiles a CCL program.

  If non-default compiler options are needed, create an instance of C8CompilerOptions with C8CompilerOptionsCreate. Use the C8CompilerOptionsSet... methods to modify the options then pass the options to the C8Compile call and destroy them afterwards with the C8CompilerOptionsDestroy call. In general, the calls to manipulate compiler options match the compiler options as shown by executing **c8_compiler --help** and described in the Sybase CEP documentation.

  Note: This function only complies; it does not start executing the compiled code.

  Parameters:

  - i_input_file: The CCL project file to compile. Includes the path as well as the complete file name. The function does not assume a particular filename extension such as `.ccp` or `.ccl`.
  - i_output_file : Where to write the compiled ccx program. Includes the path as well as the complete file name. The function does not assume a particular filename extension such as .ccx. Specify a location at which the c8_client program or Sybase CEP Studio

can find the file. All relative paths are relative to the current working directory of the process that calls the `C8Compile` function.

- i_workspace: The workspace name.
- i_load_name: The load name for the module. This name should be unique within the workspace. For a description of the load name, see the description of the "--name=progname" c8_compiler option in the first table in *Compile a Project or a Schema File* on page 71.
- i_opts: The compiler options to use, or NULL for defaults.

Returns: C8_OK if successful, C8_FAIL otherwise.

- **C8Status C8RemoteCompile(const C8Char * i_input_file, const C8Char * i_output_file, const C8Char * i_compiler_flags, const C8Char * i_workspace, const C8Char * i_load_name, const C8CompilerUri * i_compiler_uri); –** Purpose: Compiles a CCL program. Compilation is done on the server, not the client. For more information about server-side vs. client-side compilation, see *Client-side vs. Server-side Compilation* on page 30.

  If non-default compiler options are needed, create an instance of C8CompilerOptions with C8CompilerOptionsCreate use the C8CompilerOptionsSet... methods to modify the options, pass the options to the C8Compile call, and destroy them afterwards with the C8CompilerOptionsDestroy call. In general, the calls to manipulate compiler options match the compiler options as shown by executing **c8_compiler --help** and described in the Sybase CEP documentation.

  Note: This function only complies; it does not start executing the compiled code.

  Parameters:

  - i_input_file: The CCL project file to compile. This may include the path as well as the complete file name. The function does not assume a particular filename extension such as `.ccp` or `.ccl`. This is the path and name of a file on the client, not the server. The client reads the file and sends the text to the server to be compiled.
  - i_output_file: Where to write the compiled ccx program. This should include the path as well as the complete file name. The function does not assume a particular filename extension such as `.ccx`. You should specify a location at which the c8_client program (or Sybase CEP Studio) can find the file. All relative paths are relative to the current working directory of the process that calls the C8Compile() function. Note that this is the path and name of a file on the client, not the server.
  - i_compiler_flags: The compiler options to use, or NULL for defaults.
  - i_workspac: The workspace name.
  - i_load_name: The load name for the module. This name should be unique within the workspace. For a description of the load name, see the description of the "--name=progname" c8_compiler option in the first table in *Compile a Project or a Schema File* on page 71.

- i_compiler_uri: The string containing the URI of the compiler service. Since the URI is being compiled on a remote server, this string contains the URI of that remote server in the usual form: http://hostname:port. For example, "http://Mgr1:6789".

Returns: C8_OK if successful, C8_FAIL otherwise.

- **C8Status C8CompileStreamSchema(const C8Char * i_input_file, const C8Char * i_output_file); –** Purpose: This function allows you to compile a stream schema file and create a file that contains a Tuple Descriptor.

  Parameters:

  - i_input_file: The name (optionally including the path) of the input file, or schema file. By convention, the filename extension is normally `.ccs`.
  - i_output_file: The name (optionally including the path) of the output file, or schema file. By convention, the filename extension is normally `.ccx`.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8CompilerOptions * C8CompilerOptionsCreate(); –** Purpose: Creates a newly allocated compiler options structure containing default values. The structure must be de-allocated with `C8DestroyCompilerOptions()`.

  Parameters: None.

  Returns: The pointer to the new structure or NULL if the structure could not be allocated.
- **void C8CompilerOptionsDestroy(C8CompilerOptions * i_opts); –** Purpose: Destroys an instance of the C8CompilerOptions structure.

  Parameters:

  - i_opts: The pointer to a C8CompilerOptions structure.

  Returns: Nothing.
- **C8Status C8CompilerOptionsGetRepositoryPath (const C8CompilerOptions * i_opts, const C8Char ** o_res); –** Purpose: This gets the path to the SybaseC8Repository.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - o_res: The location to write the result (the repository path) to.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Status C8CompilerOptionsSetRepositoryPath (C8CompilerOptions * i_opts, const C8Char * value); –** Purpose: Sets the repository path.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - value: The new path.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8CompilerOptionsGetDebugMode(const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: This gets a value that indicates whether the compiler options are set for debug mode.

  Parameters:

  - i_opts: The valid pointer to a compiler option's structure.
  - o_res: The location to write the result to.

  Returns: C8_TRUE if debug mode is set; C8_FALSE otherwise.
- **C8Status C8CompilerOptionsSetDebugMode(C8CompilerOptions * i_opts, C8Bool value); –** Purpose: This specifies whether to compile in debug mode.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - value: C8_TRUE to set debug mode, C8_FALSE to clear debug mode.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsGetPlaybackRate (const C8CompilerOptions * i_opts, C8Float * o_res); –** Purpose: Gets the accelerated playback rate. For example, 10.0 if the data is accelerated by a factor of 10, or 0.5 if the data is processed at 1/2 the speed indicated by the row timestamps.

  Parameters:

  - i_opts: Is a valid pointer to a compiler options structure.
  - o_res: The location to write the result to.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsSetPlaybackRate (C8CompilerOptions * i_opts, C8Float value); –** Purpose: This function sets the accelerated playback rate. For example, to process the data 10.0 times as fast, set the value to 10.0. To slow the data by a factor of 2, use 0.5.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - value: The new playback rate.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsGetDeprecationWarningsEnabled (const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: This gets a value that indicates whether deprecation warnings are enabled.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.

- o_res: The location to write the result to.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8CompilerOptionsSetDeprecationWarningsEnabled (C8CompilerOptions * i_opts, C8Bool value); –** Purpose: Specifies whether to enable or disable deprecation warnings.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - value: C8_TRUE to enable, C8_FALSE to disable.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsGetWarnImplicitConversionsEnabled (const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: Gets a value that indicates whether implicit conversion warnings are enabled.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - o_res: The location to write the result to.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsSetWarnImplicitConversionsEnabled (C8CompilerOptions * i_opts, C8Bool value); –** Purpose: This function sets whether to enable or disable implicit conversions options.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - value: C8_TRUE to enable, C8_FALSE to disable.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsGetFilterAsap (const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: Gets whether the "filter as soon as possible" optimization is enabled.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - o_res: The location to write the result to.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsSetFilterAsap (C8CompilerOptions * i_opts, C8Bool value); –** Purpose: The function specifies whether to enable the "filter as soon as possible" optimization.

  Parameters:

  - i_opts: Valid pointer to a compiler options structure.

- value: C8_TRUE to enable, C8_FALSE to disable.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8CompilerOptionsGetFoldRedundantPrimitives (const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: The function gets whether the optimization to fold redundant primitives is enabled.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - o_res: The location to write the result to.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsSetFoldRedundantPrimitives (C8CompilerOptions * i_opts, C8Bool value); –** Purpose: The function sets whether to enable the optimization to fold redundant primitives.

  Parameters:

  - i_opts: The pointed to a C8CompilerOptions structure.
  - value: C8_TRUE to enable, C8_FALSE to disable.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsGetIndexWarningsEnabled (const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: The function gets a value that indicates whether the index warnings are enabled.

  Parameters:

  - i_opts: Valid pointer to a compiler options structure.
  - o_res: Location to write the result to.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsSetIndexWarningsEnabled (C8CompilerOptions * i_opts, C8Bool value); –** Purpose: The function sets whether to enable index warnings.

  Parameters:

  - i_opts: Valid pointer to a compiler options structure.
  - value: C8_TRUE to enable, C8_FALSE to disable.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8CompilerOptionsGetShortcutAndOr (const C8CompilerOptions * i_opts, C8Bool * o_res); –** Purpose: The function gets whether "Shortcut And-Or" optimization is enabled.

  Parameters:

  - i_opts: Valid pointer to a compiler options structure.

- o_res: Location to write the result to.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8CompilerOptionsSetShortcutAndOr (C8CompilerOptions * i_opts, C8Bool value); –** Purpose: The function sets whether to enable "Shortcut And-Or" optimization.

  Parameters:

  - i_opts: Valid pointer to a compiler options structure.
  - value: C8_TRUE to enable, C8_FALSE to disable.

  Returns: C8_OK on success; C8_Fail otherwise.

Sample C Program to Compile a Project

The program below is used to compile a project.

```
...
#include "c8client.h"
#include "c8compiler.h"
int main(int argc, char **argv)
{
    /*
     * This is an example of using the Sybase CEP C API
     * to compile CCL programs.
     */
    int ret = 0;
    // The value (C8_OK or C8_FAIL) returned by C8Compile()
    int r;
    const char * ccl = "pass-through.ccl";
    const char * ccx = "pass-through.ccx";
    const char * workspace = "Default";
    C8CompilerOptions * opts = 0;
    /* You must initialize the client library before calling
     * C8Compile() or any other methods from the Client SDK!
     */
    if (C8_OK != C8ClientSDKInitialize(argv[0], 0)) {
        return -1;
    }
    /* get default compiler options. */
    opts = C8CompilerOptionsCreate();
    /* To demonstrate how to set compiler options,
     * let's tell the compiler to compile in release mode.
     * To use default compiler options, it is safe to
     * just pass NULL.
     */
    C8CompilerOptionsSetDebugMode(opts, C8_FALSE);
    r = C8Compile(ccl,        // Input file name (e.g. foo.ccl)
                  ccx,        // Output file name (e.g. foo.ccx)
                  workspace,  // Name of workspace foo will run in.
                  "",         // No need to load this under a name
                              //    other than foo.
                  opts        // Compiler options
                  );
```

```
    if (r = C8_OK) {
        ret = 0;
        printf ("Compilation successful!\n");
    } else {
        /* try printing out the actual compiler error */
        C8SizeType l_errtxtlen = C8ErrorGetMessageLength();
        ret = -1;
        printf ("Compilation failed.\n");
        if (l_errtxtlen > 0) {
            C8Char * l_errbuf = (C8Char *) C8Malloc(l_errtxtlen);
            C8ErrorGetMessageText(l_errbuf, l_errtxtlen);
            printf("Error message: %s\n", l_errbuf);
            C8Free(l_errbuf);
        }
    }
    /* don't forget to free the compiler options */
    if (opts)
        C8CompilerOptionsDestroy(opts);
    /* and shut down the library */
    if (C8ClientSDKShutdown() != C8_OK) {
        /* could not shut down the library */
        ret = -1;
    }
    return ret;
}
```

To compile this, follow the instructions that were given for compiling the "register query" example at the end of *Example* on page 202.

Additional Sources of Information

The example program `example_ccl_compile.c` is included in the SDK examples.

On Microsoft Windows, if you installed to the default directory, the file is in:

```
C:\Program Files\sybasec8\server\sdk\c\examples
```

On UNIX-like operating systems, if you installed to the default directory, the file is in:

```
/home/<userid>/sybasec8/server/sdk/c/examples
```

Start/Stop a Project

This section introduces start and stop project functions, their purposes, and parameters.

- **C8Status C8StartProgram (cost C8Char *i_mgr_uri, const C8Char *i_ws_name, const C8Char *i_filename); –** Purpose: Starts a compiled program in Sybase CEP Server. This does not compile the program; you must do the compilation beforehand.

  Parameters:

  - i_mgr_uri: The URI of the manager.
  - i_ws_name: The name of the workspace.

- i_filename: The name of the compiled .ccx file.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StartProgramA (const C8Char *i_mgr_uri, const C8Char *i_ws_name, const C8Char *i_filename, const C8UserCredentials *i_credentials); –** Purpose: Starts a compiled program in Sybase CEP Server. This does not compile the program; you must have already done the compilation separately.

  This version is used if the User Authentication feature limits the right to start and stop programs.

  Parameters:

  - i_mgr_uri: The URI of the manager.
  - i_ws_name: The name of the workspace.
  - i_filename: The name of the compiled .ccx file.
  - i_credentials: A pointer to a structure containing the user's credentials (username and password).

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StopProgram (const C8Char *i_mgr_uri, const C8Char *i_ws_name, const C8Char *i_progname); –** Purpose: Stops a running program in Sybase CEP Server.

  Parameters:

  - i_mgr_uri: The URI of the manager.
  - i_ws_name: The name of the workspace.
  - i_progname: The name of the program. If you compiled the program yourself, it is the load name.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StopProgramA (const C8Char *i_mgr_uri, const C8Char *i_ws_name, const C8Char *i_progname, const C8UserCredentials *i_credentials); –** Purpose: Stops a running program in Sybase CEP Server. This version is used if the User Authentication feature limits the right to start and stop programs.

  Parameters:

  - i_mgr_uri: The URI of the manager.
  - i_ws_name: The name of the workspace.
  - i_progname: The name of the program. If you compiled the program yourself, it is the load name.
  - i_credentials: A pointer to a structure containing the user's credentials (username and password).

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8GetStreamSchema (const C8Char *i_uri); –** Purpose: Gets the schema of the stream.

Parameters:

- i_uri: The URI of the stream.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8GetStreamSchemaA (const C8Char *i_uri, const C8UserCredentials *i_credentials); –** Purpose: Gets the schema of the stream. This version is used if the User Authentication feature limits the right to start and stop programs.

  Parameters:

  - i_uri: The URI of the stream.
  - i_credentials: A pointer to a structure containing the user's credentials (username and password).

  Returns: C8_OK on success; C8_Fail otherwise.

Monitoring Servers and Queries

This section provides information about monitoring servers and queries.

Status API

This section describes the API for getting status information, including information about a Manager server, a workspace, or a project.

- **C8StatusInfo * C8GetManagerStatusInfo(const C8Char * i_manager_uri); –** Purpose: Gets the status of a manager.

  Parameters:

  - i_manager_uri: The manager URI.

  Returns: The pointer to the status info object, or NULL if the function fails. The returned object must be freed with `C8StatusInfoDestroy()`.

- **C8StatusInfo * C8GetManagerStatusInfoA(const C8Char * i_manager_uri, const C8UserCredentials *i_credentials); –** Purpose: Gets the status of a manager. This version is used if the User Authentication feature limits the right to get status.

  Parameters:

  - i_manager_uri: The manager URI.
  - i_credentials: The pointer to a structure containing the user's credentials (username and password).

  Returns: The pointer to the status info object, or NULL if the function fails. The returned object must be freed with `C8StatusInfoDestroy()`.

- **C8StatusInfo * C8GetWorkspaceStatusInfo(const C8Char * i_manager_uri, const C8Char * i_workspace_name, C8Bool i_need_ccx_info); –** Purpose: Gets the status of a workspace.

  Parameters:

  - i_manager_uri: The manager URI.
  - i_workspace_name: The workspace name.
  - i_need_ccx_info: Whether to include CCX information in the returned status object.

  Returns: The pointer to the status info object, or NULL if the function fails. The returned object must be freed with `C8StatusInfoDestroy()`.
- **C8StatusInfo * C8GetWorkspaceStatusInfoA(const C8Char * i_manager_uri, const C8Char * i_workspace_name, C8Bool i_need_ccx_info, const C8UserCredentials *i_credentials); –** Purpose: Gets the status of a workspace.

  This version is used if the User Authentication feature limits the right to get status.

  Parameters:

  - i_manager_uri: The manager URI.
  - i_workspace_name: The workspace name.
  - i_need_ccx_info: Whether to include CCX information in the returned status object.
  - i_credentials: The pointer to a structure containing the user's credentials (username and password).

  Returns: The pointer to the status info object, or NULL if the function fails. The returned object must be freed with `C8StatusInfoDestroy()`.
- **C8StatusInfo * C8GetApplicationStatusInfo(const C8Char * i_manager_uri, const C8Char * i_workspace_name, const C8Char * i_app_name, C8Bool i_need_ccx_info); –** Purpose: Gets the status of a CCL application.

  Parameters:

  - i_manager_uri: The manager URI.
  - i_workspace_name: The workspace name.
  - i_app_name: The CCL application name.
  - i_need_ccx_info: Whether to include CCX information in the returned status object.

  Returns: The pointer to the status info object, or NULL if the function fails. The returned object must be freed with `C8StatusInfoDestroy()`.
- **C8StatusInfo * C8GetApplicationStatusInfoA(const C8Char * i_manager_uri, const C8Char * i_workspace_name, const C8Char * i_app_name, C8Bool i_need_ccx_info, const C8UserCredentials *i_credentials); –** Purpose: Gets the status of a CCL application. This version is used if the User Authentication feature limits the right to get status.

  Parameters:

- i_manager_uri: The manager URI.
- i_workspace_name: The workspace name.
- i_app_name: The CCL application name.
- i_need_ccx_info: Whether to include CCX information in the returned status object.
- i_credentials: The pointer to a structure containing the user's credentials (username and password).

Returns: The pointer to the status info object, or NULL if the function fails. The returned object must be freed with `C8StatusInfoDestroy()`.

- **void C8StatusInfoDestroy(C8StatusInfo * i_status); –** Purpose: Destroys a status object.

  Parameters:

  - i_status: The valid pointer to a status object.

  Returns: Nothing.
- **C8Status C8StatusInfoGetTimestamp(const C8StatusInfo * i_status, C8Timestamp * o_ts); –** Purpose: Returns the timestamp of the status info object.

  Parameters:

  - i_status: The valid pointer to a status object.
  - o_ts: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessage (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, const C8Char * i_msg_name, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_name: The name of an object in a group.
  - i_msg_name: The name of a message in an object.
  - o_msg: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageByMessageIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, C8UInt i_msg_ndx, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.

- i_grp_name: The name of a group.
- i_obj_name: The name of an object in a group.
- i_msg_ndx: The valid index of an object in a group (0-based).
- o_msg: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetMessageByObjectIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, const C8Char * i_msg_name, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_name: The name of a message in an object.
  - o_msg: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageByObjectAndMessageIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, C8UInt i_msg_ndx, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_msg: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageByGroupIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, const C8Char * i_msg_name, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_name: The name of a message in an object.
  - o_msg: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetMessageByGroupAndMessageIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, C8UInt i_msg_ndx, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_msg: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageByGroupAndObjectIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, const C8Char * i_msg_name, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx : The valid index of a group (0-based).
  - i_obj_ndx : The valid index of an object in a group (0-based).
  - i_msg_name: The name of a message in an object.
  - o_msg" Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageByGroupAndObjectAndMessageIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, C8UInt i_msg_ndx, const C8Message ** o_msg); –** Purpose: Gets a status message. The pointer to the message is valid while the pointer to the status is valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_msg " Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoHasMessage(const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, const C8Char * i_msg_name, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

Parameters:

- i_status: The valid pointer to a status object.
- i_grp_name: The name of a group.
- i_obj_name: The name of an object in a group.
- i_msg_name: The name of a message in an object.
- o_res: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoHasMessageByMessageIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, C8UInt i_msg_ndx, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_name: The name of an object in a group.
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_res: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoHasMessageByObjectIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, const C8Char * i_msg_name, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_name: The name of a message in an object.
  - o_re: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.y
- **C8Status C8StatusInfoHasMessageByObjectAndMessageIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, C8UInt i_msg_ndx, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).

- o_res: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoHasMessageByGroupIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, const C8Char * i_msg_name, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_name: The name of a message in an object.
  - o_res: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoHasMessageByGroupAndMessageIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, C8UInt i_msg_ndx, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_res: Location where the result is to be written.

  Returns:C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoHasMessageByGroupAndObjectIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, const C8Char * i_msg_name, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_name: The name of a message in an object.
  - o_res: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoHasMessageByGroupAndObjectAndMessageIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, C8UInt i_msg_ndx, C8Bool* o_res); –** Purpose: Checks whether the status info object contains a message.

  Parameters:

- i_status: The valid pointer to a status object.
- i_grp_ndx: The valid index of a group (0-based).
- i_obj_ndx: The valid index of an object in a group (0-based).
- i_msg_ndx: The valid index of an object in a group (0-based).
- o_res: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetValue (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, const C8Char * i_msg_name, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_name: The name of an object in a group.
  - i_msg_name: The name of a message in an object.
  - o_val: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetValueByMessageIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, C8UInt i_msg_ndx, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_name: The name of an object in a group.
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_val: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetValueByObjectIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, const C8Char * i_msg_name, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_name: The name of a message in an object.

- o_val: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetValueByObjectAndMessageIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, C8UInt i_msg_ndx, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_val: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetValueByGroupIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, const C8Char * i_msg_name, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_name: The name of a message in an object.
  - o_val: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetValueByGroupAndMessageIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, C8UInt i_msg_ndx, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_val: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetValueByGroupAndObjectIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, const C8Char * i_msg_name, const**

C8Char ** o_val); – Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

Parameters:

- i_status: The valid pointer to a status object.
- i_grp_ndx: The valid index of a group (0-based).
- i_obj_ndx: The valid index of an object in a group (0-based).
- i_msg_name: The name of a message in an object.
- o_val: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetValueByGroupAndObjectAndMessageIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, C8UInt i_msg_ndx, const C8Char ** o_val); –** Purpose: Get a string value of the message. The returned pointer is valid while the pointer to the status info object remains valid.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_val: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetGroupCount(const C8StatusInfo * i_status, C8UInt * o_cnt); –** Purpose: Get the number of groups in the status info object.

  Parameters:

  - i_status: The valid pointer to a status object.
  - o_cnt: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetGroupName(const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char ** o_grp_name); –** Purpose: Get the name of a group in the status object.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - o_grp_name: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetObjectCount (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt * o_cnt); –** Purpose: Get the number of objects in a group.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - o_cnt: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise
- **C8Status C8StatusInfoGetObjectCountByGroupIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt * o_cnt); –** Purpose: Get the number of objects in a group.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - o_cnt: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetObjectName (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, const C8Char ** o_name); –** Purpose: Get the name of an object in a group.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - o_name: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetObjectNameByGroupIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, const C8Char ** o_name); –** Purpose: Get the name of an object in a group.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - o_name: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageCount (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, C8UInt * o_cnt); –** Purpose: Get the number of messages in an object.

Parameters:

- i_status: The valid pointer to a status object.
- i_grp_name: The name of a group.
- i_obj_name: The name of an object in a group.
- o_cnt location: The location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

- **C8Status C8StatusInfoGetMessageCountByObjectIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, C8UInt * o_cnt); –** Purpose: Get the number of messages in an object.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - o_cnt: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageCountByGroupIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, C8UInt * o_cnt); –** Purpose: Get the number of messages in an object.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_nd: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - o_cnt: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageCountByGroupAndObjectIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, C8UInt * o_cnt); –** Purpose: Get the number of messages in an object.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - o_cnt: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageName (const C8StatusInfo * i_status, const C8Char * i_grp_name, const C8Char * i_obj_name, C8UInt i_msg_ndx, const C8Char ** o_name); –** Purpose: Get the name of a message.

Parameters:

- i_status: The valid pointer to a status object.
- i_grp_name: The name of a group.
- i_obj_name: The name of an object in a group.
- i_msg_ndx: The valid index of an object in a group (0-based).
- o_name: Location where the result is to be written.

Returns C8_OK on success; C8_Fail otherwise.:

- **C8Status C8StatusInfoGetMessageNameByObjectIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, C8UInt i_msg_ndx, const C8Char ** o_name); –** Purpose: Get the name of a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_name: The name of a group.
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_name: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageNameByGroupIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, C8UInt i_msg_ndx, const C8Char ** o_name); –** Purpose: Get the name of a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_name: The name of an object in a group.
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_name: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8StatusInfoGetMessageNameByGroupAndObjectIndex(const C8StatusInfo * i_status, C8UInt i_grp_ndx, C8UInt i_obj_ndx, C8UInt i_msg_ndx, const C8Char ** o_name); –** Purpose: Get the name of a message.

  Parameters:

  - i_status: The valid pointer to a status object.
  - i_grp_ndx: The valid index of a group (0-based).
  - i_obj_ndx: The valid index of an object in a group (0-based).
  - i_msg_ndx: The valid index of an object in a group (0-based).
  - o_name: Location where the result is to be written.

Returns: C8_OK on success; C8_Fail otherwise.

Tracer Message API

A stream can contain normal data messages and tracer messages, which you can use to calculate the time required for a message to move through .

Tracer messages are identified by the having message type C8_MESSAGE_TRACER. For a brief description of different message types, see *Message API* on page 121.

The Sybase CEP SDK includes functions that allow you to get and set the values of fields in these tracer messages:

- CreationUri: The value should be a valid Sybase CEP HTTP Stream URI, for example, `http://<host>:<port>/Stream/<ws-name>/<program-name>[/ <module-name> ]/<stream-name>` .
- CreationTime: The time at which the tracer message was created.
- CreationFrequency: The frequency or interval (measured in microseconds) between tracer messages with which tracer messages are injected into the stream. If there are no periodic tracers, the frequency is 0. If a tracer source is disconnected, the frequency is -1.

Below is a description of each of the tracer-related functions in the API:

- **C8Status C8TracerGetCreationURI(const C8Message* i_msg, const C8Char ** o_res); –** Purpose: Gets the creation URI field from a tracer message. The value is represented by a null-terminated string. The strings returned by this function must be freed with the `C8Free()` function.

  Parameters:

  - i_msg: The message from which you wish to read the Creation URI. The message must be of type C8_MESSAGE_TRACER.
  - o_res: Location where the result is to be written.

  Returns: C8_OK on success; C8_Fail otherwise.
- **C8Status C8TracerGetCreationTime(const C8Message* i_msg, C8Timestamp* o_res); –** Purpose: Gets the creation time from a tracer message.

  Parameters:

  - i_msg: A valid pointer to a message of type C8_MESSAGE_TRACER.
  - o_res: Location filled with the creation time on success.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8TracerGetCreationFrequency(const C8Message* i_msg, C8Interval* o_res); –** Purpose: Gets the creation frequency from a tracer message.

  Parameters:

- i_msg: A valid pointer to a message of type C8_MESSAGE_TRACER.
- o_res: Location filled with the creation frequency on success.

Returns: C8_OK on success, C8_FAIL otherwise.

- **C8Status C8TracerSetCreationURI(C8Message* i_msg, const C8Char * i_val); –** Purpose: Sets the creation URI field of a tracer message.

  Parameters:

  - i_msg: A valid pointer to a message of type C8_MESSAGE_TRACER.
  - i_val: Pointer to NULL terminated string containing the creation URI

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8TracerSetCreationTime(C8Message* i_msg, C8Timestamp i_val); –** Purpose: Sets the creation time of a tracer message.

  Parameters:

  - i_msg: A valid pointer to a message of type C8_MESSAGE_TRACER.
  - i_val: Creation timestamp.

  Returns: C8_OK on success, C8_FAIL otherwise.
- **C8Status C8TracerSetCreationFrequency(C8Message* i_msg, const C8Interval i_val); –** Purpose: Sets the creation frequency of a tracer message.

  Parameters:

  - i_msg: A valid pointer to a message of type C8_MESSAGE_TRACER.
  - i_val: Creation frequency.

  Returns: C8_OK on success, C8_FAIL otherwise.

User-Defined Functions

This chapter provides an overview of UDFs in C/C++.

User Defined Functions in C/C++

Sybase CEPallows you to write your own UDFs in C or C++ and call those functions from within CCL code.

For example, you can write a function "foo" that processes an integer value and returns a single value, and then call that function from a query:

```
INSERT INTO OutputStream
SELECT foo(InputStream.MyIntegerColumn)
FROM InputStream... ;
```

UDFs: Requirements and Example

This section describes requirements for User-Defined Functions, including supported languages and implementation.

The server must be able to find the library that contains your UDF, load the library into memory, call your UDF, and pass appropriate parameters to the UDF. Because of this, each UDF must meet the following requirements:

- The UDF must be written in C or C++. The function names that are externally visible must use C naming conventions. C++ name mangling will prevent the code from being accessible.
- The UDF implementation must be compiled into a shared library (.so or .dll) for the specific platform. That library can contain one or more user-defined functions.
- Before the server can call a function, the server must be told the name of the library, the names of the UDFs in that library, and the data types of the parameters passed to each function. This information must be stored in an XML file that the server reads when it starts up. The format of this file is described in *UDFs: XML Signatures* on page 242.
- The UDF must declare appropriate C-language data types that match the CCL data types of the parameters. For example, if the server calls the UDF with a value of type TIMESTAMP, the corresponding C-language variable must be of type C8Timestamp.

To write a UDF, you must create the following files:

- A C-language file that contains your UDF and code to pack and unpack values that are stored in a context parameter. (This is described in more detail in *Accessing Parameter Values* on page 248.)
- An XML file with the library name, function name, and parameter types.

Each of these files is described in this document.

UDFs: Packing and Unpacking Parameter Values

In addition to writing the UDF that does the work that you want, you must also write code to pack and unpack values and to detect NULL values.

As we saw in the code sample earlier:

```
INSERT INTO OutputStream
SELECT foo(InputStream.MyIntegerColumn)
FROM InputStream... ;
```

the server can call the UDF with zero or more parameters. Those parameters must be a data type that CCL supports (INTEGER, FLOAT, STRING, TIMESTAMP, and so on). Although the function is called with (and returns) a CCL data type, your C-language UDF must pass (and receive) a C data type (int, double, char *, and so on).

To ensure proper conversion between CCL data types and C data types, and to identify NULL values, CCL data types are processed and then stored in a single variable-size parameter that is

passed to your UDF. This parameter is called the context parameter. Your UDF then calls library functions that extract the data and return it to you in a C-compatible format. This is called Unpacking. When you return a value to the caller, you go through the reverse process: Pack the return value by calling a special function that stores the value in a CCL-compatible format, and set a NULL flag if appropriate.

The UDF does the following:

1. Declares appropriate C-language variables.
2. Checks for NULL values and handles them appropriately.
3. Unpacks the input parameters (reads context parameter and converts the data from the CCL data types to the C data types).
4. Does the "real work" (for example, calculates pi).
5. Packs the return value (converts the output value from a C-language data type to a CCL data type and copies that data into the output portion of the context parameter).
6. Returns to the caller.

The context variable contains a copy of all the individual parameters that are intended for your UDF. The context variable also stores values that indicate whether each of the input parameters is NULL.

The context variable is declared as type C8UDF. Thus your function will declare a single parameter of type C8UDF. For example:

```
int foo(C8Udf ctx)
{
/* "Unpack" parameters from foo and store them
 * in local variables.
 */
...
}
```

An example call to this function looks like:

```
INSERT INTO OutStream
SELECT WAvg3(var1, weight1, var2, weight2, var3, weight3)...
```

Note that we used the CCL name (WAvg3), not the C name (weightedAverage3). In the CCL code, the name is not case-sensitive.

UDFs: Example

This section contains an example that creates a UDF that calculates the weighted average of 3 values.

Here is the C signature of our UDF, along with declarations of the local variables that we use:

```
/**
* Calculates a weighted average of 3 values.
* Performs (var1*wt1 + var2*wt2 + var3*wt3)/3.0
```

```
*/
void weightedAverage3(C8Udf *ctx)
 float var1;     /* first user variable */
 float weight1;  /* 2nd user var: weight for var1 */
 float var2;     /* 3rd user var */
 float weight2;  /* 4th user var */
 float var3;     /* ... */
 float weight3);
```

You must provide an xml implementation of this signature and describe the parameters that will be passed from the CCL statement. The XML looks like:

```
<UserDefinedFunctions
  Name="TestingUserDefinedFunctions"
  Type="ns1:UserDefinedFunctionType"
  xmlns="http://www.sybase.com/udf/2005/04/"
  xmlns:udf="http://www.sybase.com/udf/2005/04/"
  xmlns:ns1="http://www.sybase.com/cpx/2005/04/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
<Description>Sybase CEP User Function Definition test cases
</Description>
<Vendor>Sybase Inc.</Vendor>
<Version>1.0</Version>
<Functions>
    <!-- Name         : Name in C code                    -->
    <!-- Library      : Library that the C func is in     -->
    <!-- CclName      : Name used in call from in CCL     -->
    <!-- IsAggregator : This function is not an aggregator -->
    <Function Name="weightedAverage3"
              Library="user_function_lib"
              CclName="WAvg3">
      <Description>
       XML to describe a C function that performs a
       weighted average of 3 values.
      </Description>
      <Input>
          <Parameter Name="var1"    Type="C8Float"/>
          <Parameter Name="weight1" Type="C8Float"/>
          <Parameter Name="var2"    Type="C8Float"/>
          <Parameter Name="weight2" Type="C8Float"/>
          <Parameter Name="var3"    Type="C8Float"/>
          <Parameter Name="weight3" Type="C8Float"/>
      </Input>
      <Output>
          <Parameter Type="C8Float"/>
      </Output>
    </Function>
   </Functions>
</UserDefinedFunctions>
```

This xml file is placed in the plugins subdirectories of both the server and Studio directory. See *Compiling a UDF and Putting It in the Correct Directory* on page 256 for more details.

Please note the following points:

1. You must specify both the name of the C function and the name that will be used by CCL statements. These names are usually the same.
   a) The name of the C function is specified in the line

      Function Name="weightedAverage3"

      and is case-sensitive.
   b) The name of the function called from CCL is specified in the line

      CclName="WAvg3"

      and is not case-sensitive.
2. The "Library" is the name of the file that contains the compiled UDF code. In the example above, the library name is

   "user_function_lib"

   When Sybase CEP loads the library code, it will look for a file with this name and with an extension that is appropriate for the operating system (.lib, .dll, or .so).
3. For each parameter, the file specifies the name and the data type.
4. The data type of the return value (output parameter) is also specified.
5. This UDF is not an aggregator function, so we omitted the optional IsAggregator portion of the function specification.

   An example call to this function looks like:

```
INSERT INTO OutStream
SELECT WAvg3(var1, weight1, var2, weight2, var3, weight3)...
```

   Use the CCL name (WAvg3), not the C name (weightedAverage3). In the CCL code, the name is not case-sensitive.

   Sample C code containing the UDF is below. A file with this code is aksi included in the installation, under `SybaseC8Repository/version/examples/FeatureExamples/FunctionsAndOperators/UserDefinedFunctions`.

```
#include <stdio.h>
#include "c8udf.h"
#if defined(_MSC_VER)
// Exporting functions from dll
#if defined(user_function_lib_EXPORTS)
#define USER_FUNCTION_EXPORT __declspec( dllexport )
#else
#define USER_FUNCTION_EXPORT __declspec( dllimport )
#endif //defined(user_function_lib_EXPORTS)
#else // defined(_MSC_VER)
#define USER_FUNCTION_EXPORT
#endif // defined(_MSC_VER)
extern "C" {
USER_FUNCTION_EXPORT void weightedAverage3(C8Udf* ctx);
}; // extern "C"
```

```
/* ------------------------------------------------
* Return the weighted average of 3 float values.
* ------------------------------------------------
*/
void weightedAverage3(C8Udf* ctx)
{
  C8Float value;   /* The output value that we return. */
  C8Float var1, weight1;
  C8Float var2, weight2;
  C8Float var3, weight3;
  int i = 0;    /* Used when looping thru the NULL indicators. */
  /* If any of the input parameters are NULL, return NULL. */
  for (i = 0; i <= 5; i++)  {
      if (C8GetIsNull(ctx, (C8UInt) i) == C8_TRUE)   {
          C8SetOutputIsNull(ctx);
          return;
          }
      }
  /* "Unpack" the actual param values intended for the UDF */
  var1    = C8GetFloat(ctx, 0);
  weight1 = C8GetFloat(ctx, 1);
  var2    = C8GetFloat(ctx, 2);
  weight2 = C8GetFloat(ctx, 3);
  var3    = C8GetFloat(ctx, 4);
  weight3 = C8GetFloat(ctx, 5);
  /* Do the real work. */
  value = (var1*weight1 + var2*weight2 + var3*weight3)/3.0;
  /* "Pack" the output value into the context variable. */
  C8SetOutputFloat(ctx, value);
  return;
}
```

You can write any C function and may call other C routines. For the above example, this may involve placing the parameters in an array and using the array or any other useful technique.

Notice the various "#if" statements necessary to externalize symbols on Windows and UNIX-like operating systems.

In this example, we use a single function to do the real work (for example, return the weighted average of some values) and to pack and unpack the parameter values. If you already have a function `foo()` that you would like to use without modifying, then you can simply write an intermediate function that unpacks the values, calls `foo()`, packs the result, and returns. When you use an intermediate function the Sybase CEP Server calls the intermediate function, and the intermediate function calls your UDF. Since the Sybase CEP Server actually calls the intermediate function, the XML file that describes the UDF must contain the name of the intermediate function.

User-Defined Aggregate Functions in C/C++

For an aggregate function, the user must have a place to store their data between invocations of the function.

For example, if you write your own `SUM()` function, you must store the previous subtotal from the previous call to the function and then add the new value from the current call to the function.

Since an aggregate function can be called by many queries at overlapping times, you cannot use local storage space or global variables to store values such as subtotals.

TYour aggregate function must work with the server to store data in a place that will persist after the UDF returns from its call. Sybase CEP provides a pair of functions that allow you to store and retrieve data. The function

```
void C8SetState(C8Udf *ctx, const void *data, C8UInt data_size)
```

allows you to pass a sequence of bytes to the server and store those bytes. The complementary function

```
const void *C8GetState(C8Udf *ctx, C8UInt *data_size)
```

allows your UDF to retrieve bytes that it stored previously. The bytes stored and retrieved are specific to a particular occurrence of a particular function in a particular statement. If your Query Module has the following statements:

```
...
SELECT aggregate_foo(col1)
FROM Window1
...
SELECT aggregate_foo(col2)
FROM Window2
```

then each aggregate_foo will have a unique internal identifier that the Server uses as part of the context variable passed to your function. The context variable then allows the server to know which aggregate_foo's bytes to return when the UDF calls

```
C8GetState(ctx, ...)
```

The `C8GetState()` and `C8SetState()` functions are documented in more detail later in this chapter.

An aggregate function operates on a window. Your UDF must take into account not only newly arriving rows, but also expiring rows. To do this, your aggregate UDF will usually be called twice each time that a new record arrives. Your aggregate UDF will be called once with the value of the new record and once with the value of the expired record. Inside your aggregate UDF, you will distinguish between the new and displaced values by calling a function named `C8IsPositiveMessage()` or `C8IsNegativeMessage()`.

Below is a sample of code that shows typical usage of these aggregate-related functions. The sample shows an aggregate UDF that performs the same work as the pre-defined `AVG()` function.

```
...
typedef struct _AvgData {
    C8Int m_sum;
    int m_count;
} AvgData;
...
AvgData initial_data = { 0, 0 };
C8UInt size = 0;
struct AvgData *data_ptr = NULL;
/* Get state (if any) from previous calls. */
data_ptr = (AvgData*)C8GetState(ctx, &size);
// If there is no state from previous calls, then
// this is probably the first invocation and we must
// allocate memory.
if (data_ptr == NULL || (size != sizeof(AvgData))) {
    data_ptr = &initial_data;
}
/* Update the sum and count */
if(C8IsPositiveMessage(ctx)) {
    data_ptr->m_sum += C8GetInt(ctx, 0);
    data_ptr->m_count++;
} else {
/* Negative message - a row just exited the window */
    data_ptr->m_sum -= C8GetInt(ctx, 0);
    data_ptr->m_count--;
}
/* Set the result */
C8SetOutputFloat(ctx,
(C8Float)(data_ptr->m_sum)/data_ptr->m_count);
/* Save state */
C8SetState(ctx, data_ptr, sizeof(AvgData));
return;
...
```

The initial_data variable is allocated as a local, non-static variable. This is the variable that the function uses to store data if this is the first time that the function has been called.

When data is passed to the `C8SetState()` function, `C8SetState()` copies that data from the user's local memory (for example, init_data) to persistent memory that the server allocates. The storage allocated locally (for example, init_data) does not itself persist after the UDF returns to the caller. Only a copy of the contents persists. The UDF is responsible for deallocating any memory that the UDF allocated, and the Server is responsible for deallocating any memory that the Server allocated. Do not try to deallocate any memory (such as the memory returned by `C8GetState`) that was allocated by the server.

You must also set the IsAggregator attribute to "true" in the XML file that describes the function.

You may want to look at the sample code for the `runningAverage()` function, located in the SybaseC8Repository, under `examples/FeatureExamples/FunctionsAndOperators/UserDefinedFunctions/src`.

UDFs: XML Signatures

This section describes the XML signature file and where to store it.

software requires the C function description to be in XML. The XML is stored in a file with the extension .udf, and a copy of that file is put in the plugins directory of both the server and Studio. See *Compiling a UDF and Putting It in the Correct Directory* on page 256 for details about where to put this XML signature file.

This file's contents are similar to the following:

```
<!-- Header information -->
<UserDefinedFunctions
    Name="TestingUserDefinedFunctions"
    Type="ns1:UserDefinedFunctionType"
    xmlns="http://www.sybase.com/udf/2005/04/"
    xmlns:udf="http://www.sybase.com/udf/2005/04/"
    xmlns:ns1="http://www.sybase.com/cpx/2005/04/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  >
  <!-- The "Functions" section contains descriptions of 1 or
       more functions.
   -->
  <Functions>
    <!-- There is a "Functions" section for each function. -->
    <Function>
    ... info about the specific function goes here ...
    </Function>
  </Functions>
</UserDefinedFunctions>
```

This header information may change from version to version. We recommend that you copy one of the .udf files that we provide, remove all of the Function sections, and then add descriptions for one or more functions.

For each function, the translation of the C function to Sybase CEP UDF signatures is straightforward.

For the example below:

```
1. void weightedAverage3(float var1, /* first user variable */
2.           float weight1,      /* 2nd user var: weight for var1 */
3.            float var2,         /* 3rd user var */
4.            float weight2,      /* 4th user var */
5.            float var3,         /* ... */
6.            float weight3);
```

The user-defined xml is similar to the following:

```
2.   <Functions>
3.    <Function Name="weightedAvg3"
             InitFunctionName="weightedAvg3Init"
             ShutdownFunctionName="weightedAvg3Shutdown"
             Library="user_function_lib"
             CclName="WAvg3"
             IsAggregator="false">
4.       <Input>
5.          <Parameter Name="var1"    Type="C8Float"/>
6.          <Parameter Name="weight1" Type="C8Float"/>
7.          <Parameter Name="var2"    Type="C8Float"/>
8.          <Parameter Name="weight2" Type="C8Float"/>
9.          <Parameter Name="var3"    Type="C8Float"/>
10.         <Parameter Name="weight3" Type="C8Float"/>
11.      </Input>
12.      <Output>
13.        <Parameter Name="result" Type="C8Float"/>
14.      </Output>
15.   </Function>
16.  </Functions>
```

For each function described in the .udf file, the initial function element is always named Function and has the following attributes:

- Name: Provides the name of the function that will be called from the user-designated Library. This name is case-sensitive. If you set session state in an initialization function, use `C8GetSessionState()` to retrieve it.
- InitFunctionName (optional): Identifies a function that will be called on initialization. Use the initialization function to perform resource-intensive tasks, like reading from disk, and to set session state for calls to the function identified with the Name attribute.
- ShutdownFunctionName (optional): Identifies a function that will be called on shutdown. If you have set session state in an initialization function, you must destroy the session state in the shutdown function.
- Library: The .dll containing the user function. For UNIX-like operating systems, this will be a .so libraryand the library name should not be prefaced with "lib". Omit the extensions .dll or .so. You may use the same XML file for both UNIX-like operating systems and Microsoft Windows environments without changing the library name.
- CclName: The name that will be used in CCL queries (for example, "select FOO() ... from ..."). The CclName is case insensitive.
- IsAggregator: Set to true for aggregate functions and false for other functions.

The Functions element of the XML file can describe any number of functions. This convenience allows you to group related functions and/or library modules. There can be any number of function modules in a library. In the XML file, each function should be described inside its own "<Function> ... </Function>" element. For example:

```
<Functions>
   <Function Name="foo" ...>
   ...
```

```
   </Function>
   <Function Name="bar" ...>
   ...
   </Function>
</Functions>
```

The Input element inside each Function element must contain a list of all input parameters. Only one Input element is permitted per function.

Each Parameter element inside the Input element includes two attributes: the parameter Name and Type. The Name is arbitrary but should reflect the parameter usage because it is used for error reporting. If there is a type mismatch for any parameters at runtime, these names are passed to you.

The software uses strong type checking to help ensure correctness of data at runtime. The Type of each parameter supports this requirement. The types available map directly into C base types via these typedefs:

```
typedef int                C8Bool;
typedef char               C8Char;
typedef char*              C8CharPtr;
typedef void*              C8BlobPtr;
typedef void               C8Blob;
// For Microsoft Windows
#if defined(_MSC_VER)
typedef __int32            C8Int;
typedef unsigned __int32   C8UInt;
typedef __int64            C8Long;
typedef unsigned __int64   C8ULong;
typedef double             C8Float;
typedef __int64            C8Timestamp;
typedef __int64            C8Interval;
typedef size_t             C8SizeType;
#else  /* defined(_MSC_VER) */
typedef int32_t            C8Int;
typedef u_int32_t          C8UInt;
typedef int64_t            C8Long;
typedef u_int64_t          C8ULong;
typedef double             C8Float;
typedef int64_t            C8Timestamp;
typedef int64_t            C8Interval;
typedef size_t             C8SizeType;
#endif /* defined(_MSC_VER) */
```

Only the above typedef names are supported in the Type attribute. Throughout the C code for your UDF, you should use the typedef names rather than the underlying C types.

The output element at line 12 of the sample XML file contains only a single Parameter field and describes the function output in the same manner as input parameters. The Name attribute is optional. It is not currently possible to return more than a single value from a UDF.

UDFs: Interface Code

This section introduces to the interface code function. Includes a description of the steps required to call the function.

The function will have the following signature:

```
void weightedAverage3(C8Udf *ctx);
```

You must use accessor functions to unpack the parameters in the call, and then call the actual requested function. The accessor functions are easy to use and reflect the user's parameter typing.

The context structure allows for access to type information input/output values, NULL indicators, and meta-information about the function call. You can choose to ignore meta-information, as this contains only supporting information such as parameter names and types, but you must, extract the parameters from the input parameter list. Depending on the application, you should test each parameter for the NULL status. If a parameter is NULL, the parameter value is undefined.

Sybase CEP passes data by value as in C. Input values should not be modified or freed; they are read-only values. Memory allocated by Sybase CEP must be freed by Sybase CEP. You must free the memory you have allocated.

After your routine finishes, the output value should either contain a value or be set to NULL.

Metadata

This section introduces metadata functions, their purposes, and parameters.

- **const C8Char *C8GetFunctionName(C8Udf *ctx) –** Purpose: Returns the name of a function as defined in the UDF XML file.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: The name of a function as defined in the UDF XML file. The return value is a pointer. Do not change the memory that the pointer points to or `C8Free()` this pointer.
- **const C8Char *C8GetLibraryName(C8Udf *ctx) –** Purpose: Returns the name of the library as defined in the UDF XML file.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: The name of the library as defined in the UDF XML file. The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer.

- **const C8UInt C8GetNumberInputParameters(C8Udf *ctx) –** Purpose: Given a context, ctx, return the number of input parameters. This may be used for loops to fill arrays, error checking, and so on.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: The number of input parameters stored in the context object.
- **const C8Char *C8GetInputType(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the type string for that parameter as provided in the Parameter attribute in the UDF XML file. This may be used in error messages or type checking.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the type string for. The first parameter is 0.

  Returns: The type string for the parameter as provided in the Parameter attribute in the UDF XML file. The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer.
- **const C8_TYPES C8GetInputEnumType(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the type enumeration for that parameter. This may be used in error messages or type checking. For definitions of the valid values of the enumeration, see the definition of _C8_TYPES in the file `c8types.h`.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the type string for.

  Returns: The type string for the specified parameter as provided in the Parameter attribute in the UDF XML file.
- **const C8Char *C8GetInputName(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the name of the parameter. This may be used in error messages or type checking.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the type string for.

  Returns: The name of the parameter as provided in the Parameter attribute in the UDF XML file. Do not change the memory that the pointer points to, or `C8Free()` this pointer.

- **const C8Char *C8GetOutputType(C8Udf *ctx) –** Purpose: Provides the type enumeration for the output parameter. This may be used in error messages or type checking.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns:The type string for the output parameter. The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer.
- **const enum C8_TYPES C8GetOutputEnumType(C8Udf *ctx) –** Purpose: Provides the type enumeration for that parameter. This may be used in error messages or type checking.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: The type enumeration for that parameter.
- **const C8Char *C8GetOutputName(C8Udf *ctx) –** Purpose: Provides the string for an output parameter name. This may be used in error messages or type checking.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: The string for the output parameter name. If you have not provided an output name in the UDF XML, then this string will be empty. The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer.
- **void C8SetSessionState(C8Udf *ctx, void *data) –** Purpose: Stores the session state, which is a set of user-defined information that you want to make available across multiple invocations of the UDF's `execute()` function. Although the session state is retained by the , the user is responsible for populating and updating the session state. The session state does not contain a size because the session is not persisted over restarts. For this season, a session state should thus be created in a user-defined class allocated with the new operator or in a user-defined struct using `C8Malloc()` or a similar heap-based allocation. Session structures must not be defined on the stack because the stack is popped each time the function exits. When you are done with the session state, free it. For tips and warnings about allocating and de-allocating memory, see the section titled *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

  In the shutdown() callback function, the session state should be destroyed. Storing and retrieving session information is optional.

  Parameters:

- ctx: A context object that contains information about the parameters passed to the UDF.
- data: A pointer to the state information that you would like to be able to see the next time that the adapter's `execute()` function is called.

Returns: Nothing.

- **void * C8GetSessionState(C8Udf *ctx) –** Purpose: Returns the session state information associated with an instance of a UDF. On the initial call to `C8GetSessionState()`, a null pointer is returned. This indicates no session exists, so you are responsible for creating a session. The session state is usually constructed by a call to `C8Malloc()` and populated with a user-defined structure. Using a static or global for session states is not thread safe and the consequences are undefined.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: A pointer to session state information. The exact structure of this information is defined by you, so the function simply returns a pointer to the memory without interpreting that memory in any way.

Accessing Parameter Values

This section provides an introduction to functions that enable you to access parameter values.

- **C8Float C8GetFloat(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the floating point value of a parameter.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - nd: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: The floating point value of the specified parameter.
- **C8Int C8GetInt(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the integer value of a parameter.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: The integer value of the specified parameter.
- **C8Long C8GetLong(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the long value of a parameter.

Parameters:

- ctx: A context object that contains information about the parameters passed to the UDF.
- ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

Returns: The long value of a parameter.

- **const C8Char *C8GetCharPtr(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the C8Char pointer value of a parameter.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: The C8Char pointer value of the specified parameter. C8Char pointer corresponds to the CCL "STRING" data type. The return value of this accessor function is a pointer to a normal C string (a sequence of ASCII characters terminated with a 0 ('\0'). Note that the server (not the UDF) allocated the memory for this string, and therefore the server (not the UDF) is responsible for freeing the memory for this string. Do not modify or free this string, or change the value of the C8Char pointer (for example, to point to something else).
- **C8Timestamp C8GetTimestamp(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the timestamp value of a parameter. timestamps are in microsecond units.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: The timestamp value of the specified parameter. Sybase CEP timestamps are in microsecond units.
- **C8Interval C8GetInterval(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the interval value of a parameter. intervals are in microsecond units.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: The interval value of a parameter. Sybase CEP intervals are in microsecond units.

- **C8Bool C8GetBool(C8Udf *ctx, C8UInt ndx) –** Purpose: Provides the boolean value of a parameter.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: The boolean value of a parameter. Valid C8Boolean values (C8_TRUE and C8_FALSE) are defined in the c8udf.h file.
- **C8Bool C8GetIsNull(C8Udf *ctx, C8UInt ndx) –** Purpose: Return the NULL status for the parameter. A return value of C8_FALSE means a non-NULL value is being passed, and a value of C8_TRUE means a NULL is being passed. If a NULL is passed, the parameter value is undefined.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: C8_TRUE if a NULL is being passed; C8_FALSE if a non-NULL value is being passed.
- **const C8Blob *C8GetBlob(C8Udf *ctx, C8UInt ndx) –** Purpose: Returns a pointer to the BLOB value of a parameter. If the BLOB is NULL, the function returns 0. If the BLOB is empty, the function returns a non-zero pointer.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

  Returns: A pointer to a BLOB. The value is a "raw" BLOB (as opposed to a hex string or a base64 string). For an explanation of raw vs. hex string vs. base64 string formats, see *Data Types and Subroutines for UDFs and In-process Adapters* on page 102.

  The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer.
- **C8Char *C8GetBlobBase64String(C8Udf *ctx, C8UInt ndx) –** Purpose: Returns a pointer to that parameter's BLOB value in the form of a Base64 String. If the BLOB is NULL, the function returns 0. If the BLOB is empty, the function returns a non-zero pointer.

  Parameters:

- ctx: A context object that contains information about the parameters passed to the UDF.
- ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

Returns: A pointer to a BLOB in Base64 String format. For an explanation of raw vs. hex string vs. base64 string formats, see *Data Types and Subroutines for UDFs and In-process Adapters* on page 102.

The return value is a pointer. Do not change the memory that the pointer points to, or `C8Free()` this pointer.

- **C8UInt C8GetBlobLength(C8Udf *ctx, C8UInt ndx) –** Purpose: Returns the length of the BLOB (in bytes) of a parameter. A return value of zero indicates either that the BLOB is NULL or that the length of the BLOB is zero.

  To distinguish between these 2 cases (NULL vs. 0 length), you can do either of the following:

  - Call `C8GetIsNull()`C8GetIsNull() to determine whether the BLOB is NULL.
  - Check whether the pointer returned by `C8GetBlob()` is 0 (meaning that the BLOB is NULL) or non-zero (meaning that the BLOB is non-NULL).

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of.

  Returns: The length of the BLOB. If the parameter is not of type BLOB, the server will issue an error message and the return value of the function is undefined.
- **C8Char *C8GetBlobHexString(C8Udf *ctx, C8UInt ndx) –** Purpose: Returns a pointer to the hexadecimal representation of the BLOB value. The hexadecimal string will consist only of the characters 0 to 9 and a to f, with no line breaks or other white space. The length of the returned string is exactly 1 + 2*`C8GetBlobLength()` bytes including the string terminator.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - ndx: Indicates which parameter in the context object to get the value of.

  Returns: A pointer to a copy of the BLOB in HexString format. For an explanation of raw vs. hex string vs. base64 string formats, see *Data Types and Subroutines for UDFs and In-process Adapters* on page 102.

  You must`C8Free()`the returned pointer or a memory leak will occur.
- **C8Char *C8GetAsStringXml(C8Udf *ctx, C8UInt ndx) –** Purpose: Returns a pointer to a string containing the XML value.

Parameters:

- ctx: A context object that contains information about the parameters passed to the UDF.
- ndx: Indicates which parameter in the context object to get the value of. The first parameter is parameter 0.

Returns: A pointer to the XML field as a string.

You must`C8Free()`the returned pointer or a memory leak will occur.

After you have evaluated the function, the user should use an output accessor function to return the value to the software. The output accessor functions are:

- **void C8SetOutputInt(C8Udf *ctx, C8Int value) –** Purpose: Places the long value so that Sybase CEP can access the value.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - value: The value to place.

  Returns: Nothing.
- **void C8SetOutputLong(C8Udf *ctx, C8Long value); –** Purpose: Places a long value so that can access the value.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - value: The value to place.

  Returns: Nothing.
- **void C8SetOutputFloat(C8Udf *ctx, C8Int value); –** Purpose: Places the long value so that Sybase CEP can access the value.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - value: The value to place.

  Returns: Nothing.
- **void C8SetOutputTimestamp(C8Udf *ctx, C8Timestamp value); –** Purpose: Places the long value so that can access the value.

  Parameters:

- ctx: A context object that contains information about the parameters passed to the UDF.
- value: The value to place.

Returns: Nothing.

- **void C8SetOutputInterval(C8Udf *ctx, C8Interval value); –** Purpose: Places the long value so that Sybase CEP can access the value.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - value: The value to place.

  Returns: Nothing.
- **void C8SetOutputBool(C8Udf *ctx, C8Bool value) –** Purpose: Places the long value so that can access the value.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - value: The value to place.

  Returns: Nothing.
- **void C8SetOutputCharPtr(C8Udf *ctx, C8CharPtr value) –** Purpose: Places an output value so that Sybase CEP can access the value.

  The second parameter of this function, C8CharPtr, is a STRING. will copy the string to its own internal buffer. You can then free or delete the string as desired. For example, your code might look like:

  ```
  char *ErrorStr1 = NULL;
      ErrorStr1 = (char *) malloc(MaxErrorMsgSize);
      sprintf(ErrorStr1, "Error...", ...);
      C8SetOutputCharPtr(ctx, (C8CharPtr) ErrorStr1);
      free(ErrorStr1);
  ```

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - value: The value to place.

  Returns: Nothing.
- **void C8SetOutputBlob(C8Udf *ctx, const C8Blob *blob, const C8UInt blob_length) –** Purpose: Returns the BLOB as the output value. The BLOB value must be a byte array of blob_length bytes. The BLOB is copied to an internal Sybase CEP buffe, so you should deallocate your copy after `C8SetOutputBlob()` returns.)

Parameters:

- ctx: A context object that contains information about the parameters passed to the UDF.
- blob: The BLOB that you want to return. The BLOB must be a raw BLOB. For an explanation of raw vs. hex string vs. base64 string formats, see *Data Types and Subroutines for UDFs and In-process Adapters* on page 102.
- blob_length: The length of the BLOB in bytes.

Returns: Nothing.

- **void C8SetOutputBlobBase64String(C8Udf *ctx, const C8Char *base64_string) –** Purpose: Returns a BLOB as the output value. The BLOB is copied to an internal buffer, so you should deallocate your copy after `C8SetOutputBlobBase64String()` returns.)

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.
  - base64_string: The BLOB in Base64 String format that you want to return. For an explanation of raw vs. hex string vs. base64 string formats, see *Data Types and Subroutines for UDFs and In-process Adapters* on page 102.

  Returns: Nothing.
- **void C8SetOutputIsNull(C8Udf *ctx) –** Purpose: Sets to NULL. Any other output values are ignored if this function is used.

  Parameters:

  - ctx: A context object that contains information about the parameters passed to the UDF.

  Returns: Nothing.

Aggregator UDF functions

An introduction to functions used in aggregator UDFs, their purposes, and parameters.

- **C8Bool C8IsPositiveMessage(C8Udf *ctx) –** Purpose: Indicates that the message has entered the window (is a positive message). If so, you should add the message to the aggregation.

  Parameters:

  - ctx: A "context" object that contains information about the parameters passed to the UDF.

  Returns: C8_TRUE if the message is positive; otherwise C8_FALSE otherwise.

- **C8Bool C8IsNegativeMessage(C8Udf *ctx) –** Purpose: Indicates that the message has exited the window (is a "negative" message). If so, you should subtract the message to the aggregation.

  Parameters:

  - ctx: A "context" object that contains information about the parameters passed to the UDF.

  Returns:
- **void C8SetState(C8Udf *ctx, const void *data, C8UInt data_size) –** Purpose: Saves the user-defined data structure of data_size bytes. This function copies your data into memory allocated by the server. This memory will persist after your UDF has returned. Retrieve the saved memory by using the `C8GetState()` function.

  Parameters:

  - ctx: A "context" object that contains information about the parameters passed to the UDF.
  - data: The data that you want to save as state information.
  - data_size: The number of bytes of data you are saving.

  Returns: Nothing.
- **const void *C8GetState(C8Udf *ctx, C8UInt *data_size) –** Purpose: Obtains the current state. The return value has type void * and should be cast to the user-defined data structure and the data_size pointer will contain the number of bytes in the data structure. The return value is a pointer. You should not change the memory that the pointer points to, and you should not `C8Free()` this pointer.

  Parameters:

  - ctx: A "context" object that contains information about the parameters passed to the UDF.
  - data_size: A pointer to a location into which the function can insert the number of bytes retrieved.

  Returns: A pointer to the memory that was retrieved.

Memory Management API

The memory management API is the same for in-process and out-of-process operations.

See *Memory Management API* on page 109. Some special cautions apply when these are used with in-process operations. See *Notes about Allocating and Deallocating Memory in In-process Code* on page 110.

Compiling a UDF and Putting It in the Correct Directory

This section provides the directory locations for the files on Windows and Unix-like operating systems.

You must code the UDF in C and create a .dll file for windows or a .so file for UNIX-like operating systems. You must also describe the function signature in terms of the XML file (.udf file) detailed above.

You must copy the .udf file to both the server and Sybase CEP Studio plugins directories, as well as the server bin directory.

On Microsoft Windows, these directories are typically

```
C:\Program Files\SybaseC8\Server\bin
C:\Program Files\SybaseC8\Server\plugins
C:\Program Files\SybaseC8\Studio\plugins
```

On UNIX-like operating systems, these directories are typically

```
/home/<userid>/sybasec8/server/bin
/home/<userid>/sybasec8/server/plugins
/home/<userid>/sybasec8/studio/plugins
```

Because the Sybase CEP Server and Sybase CEP Studio load these files at startup, if the Server or Sybase CEP Studio is running when the files are copied, the server or Sybase CEP Studio must be restarted to load these files.

You must thoroughly debug the UDFs in a context other than the Sybase CEP environment since the Sybase CEP software contains no debugging information.

Sybase CEP Access Function Header and Source Files

Sybase CEP provides header files containing typedef and context structures, as well as the access functions, in C source code format.

This distribution is not intended to convey permission to modify these sources but is intended to allow compilation with your own function code. Thus, typedefs for C8Int, as well as access functions, are provided. The three context structures are provided as well as code to access these structures. You should not modify this code as the Sybase CEP Server and Sybase CEP Studio depend on these structures.

Compiling a UDF

To compile your UDF, you must specify appropriate settings for your compiler.

1. Specify that the compiler should generate a shared object file (.so) or a .DLL file.
2. Ensure your list of "include" directories includes the directory that holds `c8udf.h`c8udf.h.

If you are on Microsoft Windows and are using Microsoft's Visual Studio, please do the following:

3. Start Microsoft Visual Studio.
4. Create a project file by going to the menu and clicking on File -> New -> Project.
   a) Specify that this is a Visual C++ Project.
   b) Click on "Win32".
   c) In the right-hand pane, Click on "Win32 Project".
   d) Fill in the name that you'd like to use for your project.
   e) Browse and specify the directory in which you want the project to be stored.
   f) Click on "OK".
   g) The next window to appear is the "Win32 Application Wizard" window. On the left, click on "Application Settings", then click on "DLL".
   h) Click on "Finish".
5. Microsoft Visual Studio creates a simple .cpp file to use as a starting point. We recommend that you remove all the contents of this file and insert your own C code for the UDF. Make sure that your code includes the following:

```
#include "c8udf.h"
    // Ensure functions are "exported" properly from dll.
    #if defined(_MSC_VER)
    #define USER_FUNCTION_EXPORT __declspec( dllexport )
    #else // defined(_MSC_VER)
    #define USER_FUNCTION_EXPORT
    #endif // defined(_MSC_VER)
    extern "C" {
    USER_FUNCTION_EXPORT void my_function(C8Udf* ctx);
    }; // extern "C"
```

   You will need to customize the names and return types of each UDF and any intermediate function, if you have one.
6. If you have other C-language source files that you need, add them to the project.
7. You will need to update several settings that are available in the "Property Pages" for this project.
   a) Update the list of directories to search for include files.

      To do this, go to the menu, click on "Project" and then on "Properties". You should get a new window titled something like MySample Property Pages.

      In the left-hand pane of this window, click on "Configuration Properties", then "", and then on "General".

      The right-hand pane should now show a list of settings that you can modify.

      Click in the field to the right of Additional Include Directories and add the directory that contains the file `c8udf.h file` which is included with the Sybase CEP product.

On Microsoft Windows, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk\c\include
```

On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\SybaseC8\Server\sdk\c\include
```

You can also add other directories if necessary for your UDF.

b) Turn off precompiled headers.

To do this, go to the left-hand pane in the Property Pages window, click on "" and then on Precompiled headers, then click on Create/Use Precompiled Header and set it to Not Using Precompiled Headers.

c) Add the Sybase CEP library directory which contains accessor functions that let you read and write information to the context variable to the list of library directories. To do this, go to the left-hand pane of the Property Pages window, click on "Linker" and then on "General".

In the field to the right of "Additional Library Directories", add the directory that contains the library.

On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk\c\lib
```

On Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\SybaseC8\Server\sdk\c\lib
```

d) Tell the linker not to include debugging information. To do this, go to the left-hand pane of the Property Pages window, click on "Linker" and then on "Debugging". For the field "Generate Debugging Info", change the value to "No".

e) Add a dependency on the `c8_sdk_server_lib.lib` file. To do this, go to the left-hand pane of the Property Pages window, click on "Linker" and then on "Input".

In the field to the right of "Additional Dependencies", enter

```
c8_sdk_server_lib.lib
```

You do not need to enter the complete path; entering the file name is sufficient.

To double check that you haven't skipped a step, you can look at the "Command Line" for the C/C++ compiler and the Command Line for the Linker. This shows the command-line parameters passed from Microsoft's GUI IDE to the command-line compiler and linker.

To view the command line for the compiler, go to the left-hand pane of the Property Pages window, click on "C/C++" and then click on "Command Line". The command line should look similar to the following:

```
/Od /I "C:\Program Files\SybaseC8\Server\sdk\c\include"
  /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_USRDLL"
```

```
  /D "TEST_UDF_AGGR1_EXPORTS" /D "_UNICODE"
  /D "UNICODE" /D "_WINDLL" /Gm /EHsc /RTC1 /MDd
  /Fo"Debug\\" /Fd"Debug\vc80.pdb" /W3 /nologo /c
  /Wp64 /ZI /TP /errorReport:prompt
```

If you set the warning level to a value other than 3, the "/W3" will be different.

The command line may or may not include:

```
/D "_DEBUG"
```

If the command line includes this, you may only be able to use the .DLL on a computer that has the debug version of the C runtime library. (For more information, see the *Troubleshooting* on page 351 section below.)

To view the command line for the linker, go to the left-hand pane of the Property Pages window, click on "Linker" and then click on "Command Line". The command line should look similar to the following:

```
/OUT:"C:\c8test\E2\C_SDK\TAggr1\TAggr1\Debug\TAggr1.dll"
  /INCREMENTAL /NOLOGO
  /LIBPATH:"C:\Program Files\SybaseC8\Server\sdk\c\lib"
  /DLL /MANIFEST
  /MANIFESTFILE:"Debug\TAggr1.dll.intermediate.manifest"
  /SUBSYSTEM:WINDOWS /MACHINE:X86 /ERRORREPORT:PROMPT
  c8_sdk_server_lib.lib kernel32.lib user32.lib gdi32.lib
  winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib
  oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
```

Now that you have entered all the project properties, click the "OK" button on the "Property Pages" window. At this point, you should be ready to compile.

8. To compile, use the appropriate option on the "Build" menu, for example, "Build MySample".
9. Copy the DLL file to the server's bin directory, and copy the XML (.udf) file to both the Server's and the Studio's plugins directory. On Microsoft Windows, these directories default to:

   ```
   C:\Program Files\SybaseC8\Server\bin
       C:\Program Files\SybaseC8\Server\plugins
       C:\Program Files\SybaseC8\Studio\plugins
   ```

   If you are using a command line compiler such as cc, you can use a make file or a script file.
10. The examples below are based on the cc compiler found on many UNIX-like operating systems.
    a) This example uses a make file:

    ```
    TARGET=libc8_udf_regexp_match.so
            all: $(TARGET)
            OBJECTS = regexp_match.o regexpr2.o syntax2.o
    ```

```
        C8 = /home/henry/sybasec8
        SDK_PATH = $(C)/server/sdk/c
        INC = -I$(SDK_PATH)/include -I$(SDK_PATH)/include/
stlport
       C8_LIB = -L$(SDK_PATH)/lib -lc8_sdk_server_lib -lstlport
        CPPFLAGS = -fPIC -shared
        %.o : %.cpp
  gcc $(INC) $(CPPFLAGS) $< -o $@
        $(TARGET) : $(OBJECTS)
  gcc -o $@ $?
```

You cannot copy this and use it directly. In addition to customizing the file names, you will also have to replace blank spaces with tab characters in appropriate places to satisfy the make program.

If you are compiling for a x86 architecture, you will probably also want to use the `-m64` flag or equivalent.

b) This example uses a script:

```
cc -I${HOME}/sybasec8/server/sdk/c/include \
     -o libmma.so \
     -L${HOME}/sybasec8/server/sdk/c/lib/ \
     -lc8_sdk_server_lib \
     -fPIC \
     -shared \
     mma.cpp
```

where `mma.c` is the name of your source code file and `libmma,so` is the name you'd like to use for the library file.

`-I` specifies the directory(s) to be searched for #include files.

`-o` specifies the name of the output file. Typically, this is a shared library file.

`-L` specifies the directory(s) to search for library files that need to be linked with this one.

`-l` specifies the name of the Sybase CEP library to link with to use the functions.

`-fPIC` specifies that the compiler should generate Position-Independent Code, or code for dynamic linking.

`-shared` specifies that the output will be a shared object library file as opposed to a stand-alone executable program.

Although on UNIX-like operating systems your library file name will typically be of the form `libXYZ.so`, your .udf file should specify only "`mma`" as the library name; do not specify "`libmma.so`" in the .udf file. The .udf file interpreter will make platform-specific adjustments for the filename extension (.so vs. .dll) and, if necessary, add an initial "lib" prefix.

If you are compiling for a 64-bit x86 architecture, you will probably also want to use the `-m64` flag or equivalent.

11. Compile the code by executing the script or the make file.
12. Copy the shared library file to the server's bin directory. For example:

    ```
    /home/<userid>/sybasec8/server/bin
    ```

13. Copy the XML (.udf) file to both the Server and Studio's plugins directory:

    ```
    /home/<userid>/sybasec8/server/plugins
    ```

    ```
    /home/<userid>/sybasec8/studio/plugins
    ```

UDFs: Summary

This section describes the functions and files you must create to call a UDF.

For each UDF that you want called, you will create two functions:

- The UDF: Gets the context parameter from the server, unpacks the intended parameters from the context parameter, and then processes them.
- If you already have a function that does what you want, you can use an intermediate function that gets the context parameter from the server, unpacks the intended parameters from the context parameter, and then calls your function. This intermediate function also packs the output value into the context variable and returns an error indicator. Both functions will be put in the same library.

You will create two files:

- A library file that contains your function and the corresponding intermediate function.
- An XML file that lists the name of the library, the name of the intermediate function, and the data types of the parameters in the UDF.

UDFs: Cautions

- Because your UDF can be called from multiple queries, or multiple places in the same query, your C code should be re-entrant. Specifically, it should avoid static variables.
- Your UDF should free any memory that it allocates unless the Sybase CEP API description specifically states that the Sybase CEP library function will free the memory.
- Your UDF should not free any memory that is passed to it unless the Sybase CEP API description specifically states that the user should free the memory. For example, do not free the context variable that is passed to the UDF.
- In CCL statements, your function should be used as a per-row function, not as an aggregate function such as SUM, AVG, and so on.
- Since the server reads the library file and the XML description of the UDF at the time that the server starts, changes to the library file or XML file will not take effect until the next time that the server starts.

- Since the UDF runs as part of the server, an error in the UDF can cause the server to crash. Your UDF should be thoroughly tested before you use it in a production system.
- You should explicitly handle the following:
  - Error conditions that occur inside your own UDF.
  - NULL input values.

Querying a Public Window in C/C++

Sybase CEP Server supports public windows, which you can query from outside the server.

Public windows are created with the CREATE PUBLIC WINDOW statement. For an overview of public windows, see *Public Windows* on page 65. For syntax and other information about the CREATE PUBLIC WINDOW statement, see Sybase CEP *CCL Reference Guide* .

The API functions for querying a public window are shown below:

- **C8WindowQueryResultSet *C8QueryWindow(const char *i_mgr_uri, const char *i_workspace_name, const char *i_project_name, const char *i_sql); –** Purpose: Submits a query of a public window and returns the result set.

  Parameters:

  - i_mgr_uri: The URI of the server/manager.
  - i_workspace_name: The name of the workspace that contains the public window.
  - i_project_name: The name of the project that defines the window.
  - i_sql: A string that contains the text of the query to execute.

  Returns: The result/set or NULL. The returned result/set needs to be destroyed with `C8WindowQueryResultDestroy()` .
- **C8WindowQueryResultSet *C8QueryWindowA(const char *i_mgr_uri, const char *i_workspace_name, const char *i_project_name, const char *i_sql, const C8UserCredentials *i_credentials); –** Purpose: Submits a query of a public window and returns the result set. This function includes a parameter that allows you to specify the credentials of the user whom you want to act as. For general information about user authentication, see the *Sybase CEP Installation Guide* . For more information about user authentication and the Sybase CEP C/C++ SDK, see *User Authentication* on page 271.

  Parameters:

  - i_mgr_uri: The URI of the server/manager.
  - i_workspace_name: The name of the workspace that contains the public window.
  - i_project_name: The name of the project that defines the window.
  - i_sql: A string that contains the text of the query to execute.
  - i_credentials: Credentials (user name and password).

  Returns: The result/set or NULL.

Important: The returned result/set needs to be destroyed with `C8WindowQueryResultDestroy()` .

- **void C8WindowQueryResultDestroy(C8WindowQueryResultSet *i_result); –** Purpose: Frees the given public window query result/set.

  Parameters:

  - i_result: A valid pointer to a query result/set.

  Returns: Nothing.
- **const C8Schema *C8WindowQueryResultGetSchema(C8WindowQueryResultSet *i_result); –** Purpose: Returns the schema corresponding to the result of the public window query. The returned schema remains valid while the C8WindowQueryResultSet remains valid.

  Parameters:

  - i_result: A valid pointer to a query result/set.

  Returns: The result/set's schema if successful, NULL otherwise.
- **C8SizeType C8WindowQueryResultGetSize(C8WindowQueryResultSet *i_result); –** Purpose: Returns the number of rows in result/set.

  Parameters:

  - i_result: A valid pointer to a query result/set.

  Returns: The number of rows in the result set.
- **C8WindowQueryGetRowByPos(C8WindowQueryResultSet *i_result, C8SizeType i_pos); –** Purpose: Gets a message by position. This represents a row in query result/set.

  Parameters:

  - i_result: A valid pointer to a query result/set.
  - i_pos: A valid row number. The number should be between 0 and C8WindowQueryResultGetSize() -1 (inclusive).

  Returns: The requested message.

  The message must be deleted with `C8MessageDestroy()` .

  The following code provides an example of querying a public window:

```
/* Example: Query a public window */
...
/* Submit the query and get the result set */
C8WindowQueryResultSet *res = C8QueryWindow(
        "http://manager:12345",
        "Default",
        "MyPublicWindowProject",
        "SELECT * FROM MyPublicWindow"
        );
```

```
/* If there is a result set (res is not NULL)...*/
if (res) {
    /* Get the number of rows in the resultset */
    C8SizeType row_count = C8WindowQueryResultGetSize(res);
    /* For each row/message in the resultset ... */
    for (C8SizeType ii = 0; ii < row_count; ++ii) {
        /* Get the message out of the resultset */
        C8Message *msg = C8WindowQueryGetRowByPos(res, ii);
        /* use the message here
        ...
        */
        /* Destroy message when done using it */
        C8MessageDestroy(msg);
    }
    /* destroy result set when it's no longer needed */
    C8WindowQueryResultDestroy(res);
}
...
```

RPC Plugins

A closer look at compiling and using Sybase CEP Remote Procedure Call (RPC) plugins. The RPC plugin enables CCL statements to call remote procedures and remote functions.

The information in this section is based on the RPC plugins that Sybase CEP supplies, but the steps should be nearly identical for any RPC plugin, including RPC plugins written by customers or third parties.

Sybase CEP supplies a .dll or .so file with the HTTP and SOAP plugins, so you will only need these instructions if you need a custom RPC plugin. You can use the Sybase CEP-supplied source code as a starting point for your custom plugin if you wish. The locations of the compiled library and the source code files are listed in *Generic HTTP and SOAP Plugins* on page 52.

For more information on RPCs, see *Remote Procedure Calls, Database Queries, and Public Windows* on page 47.

RPC Plugin API

This section lists the functions in the RPC Plugin API. The header file with the function prototypes is `c8rpc.h`.

Nor every RPC plugin instance, there is a data structure that stores information about this particular instance. We refer to this data structure as an RPC context or a context. Many of the functions in the API require a pointer to this context so that the server knows which RPC plugin you are trying to operate on.

Functions for Accessing Configuration Information

An introduction to functions used to access configuration information, their purposes, and parameters.

For each RPC plugin, the c8-services.xml file has an entry that contains configuration information about that RPC plugin. These functions access that configuration information.

- **const C8Char *C8RpcGetServiceName(C8Rpc *rpcctx); –** Purpose: Returns the name of the service entry in the c8-services.xml file.

  Parameters:

  - rpcctx: A pointer to the "context" (data structure) for this particular instance of the plugin.

  Returns: A pointer to the service name.
- **const C8Char *C8RpcGetServiceParameter(C8Rpc *rpcctx, const C8Char *paramName); –** Purpose: Returns the value of a given parameter from the services file for this plugin's instance. If the parameter is not found, then NULL is returned.

  Parameters:

  - rpcctx: A pointer to the "context" (data structure) for this particular instance of the plugin.
  - paramName: A string that contains the name of the parameter for which you want the value.

  Returns: A string that contains the value of the parameter.
- **const C8Schema *C8RpcGetInputSchema(C8Rpc *rpcctx); –** Purpose: Gets the input messages schema for this instance of the plugin.

  Parameters:

  - rpcctx: A pointer to the "context" (data structure) for this particular instance of the plugin.

  Returns: A pointer to a C8Schema object that describes the schema of the input messages.
- **const C8Schema* C8RpcGetOutputSchema(C8Rpc *rpcctx); –** Purpose: Gets the output messages schema for this plugin's instance.

  Parameters:

  - rpcctx: A pointer to the "context" (data structure) for this particular instance of the plugin.

  Returns: A pointer to a C8Schema object that describes the schema of the output messages.

API Function for Publishing Messages

Purpose: Returns an output message from the plugin. This function can be called multiple times to return multiple messages.

- **C8Status C8RpcReturnMessage(C8Rpc *rpcctx, C8Message *msgptr); –** If the plugin does not want to return any messages then this function should not be called. Note that the output message schema must match the schema returned by the `C8RpcGetOutputSchema()` function.

  Parameters:

  - rpcctx: A pointer to the "context" (data structure) for this particular instance of the plugin.
  - msgptr: A pointer to an empty message that the server can "fill in".

  Returns: C8_OK on success, C8_FAIL if anything goes wrong.

API Functions for Managing Session State

The C/C++ SDK provides functions that enable you to store and retrieve session state information.

The session state of an RPC plugin is analogous to the session state of an in-process adapter. An RPC Plugin's `execute()` function is typically called many times. Some plugins need to carry over information from one invocation to the next. We refer to this carried over information as "session state". Session state information is not carried over if the server crashes or is restarted. Unlike in-process adapters, RPC plugins do not allow you to set and get persistent state.

- **void C8RpcSetSessionState(C8Rpc *rpcctx, void *state); –** Purpose: Sets the plugin's session state. The state is a contiguous piece of memory containing whatever information the plugin wants to store. When you set the state in the `initialize()` function or the `execute()` function, the server stores a copy of the pointer to this memory. When you call the `C8RpcGetSessionState()` function, the server returns that same pointer.

  Parameters:

  - rpcctx: A pointer to the context for this particular instance of the plugin.
  - state: A pointer to a user-defined structure.

  Returns: Nothing.
- **void *C8RpcGetSessionState(C8Rpc *rpcctx); –** Purpose: Gets the plugin's session state. Returns NULL if no state was set.

  Parameters:

  - rpcctx: A pointer to the context for this particular instance of the plugin.

Returns: A pointer to the session state information. See the description of the `C8RpcSetSessionState()` function for more information about this pointer.

API Function for Reading Runtime Status

Describes the purpose and parameters for the API function for reading runtime status.

- **C8Status C8RpcIsCanceled(C8Rpc *rpcctx); –** Purpose: Checks if plugin needs to cancel the current operation.

  Parameters:

  - rpcctx: A pointer to the "context" (data structure) for this particular instance of the plugin.

  Returns: C8_OK on success, C8_FAIL if anything goes wrong.

Compiling an RPC Plugin and Putting It in the Correct Directory

You must compile the RPC plugin into a .dll file for windows or a .so file for UNIX-like operating systems.

You must copy the .dll (or .so) files to the server bin directory.

On Microsoft Windows, the directory is typically:

```
C:\Program Files\SybaseC8\Server\bin
```

On UNIX-like operating systems, the directory is typically:

```
/home/<userid>/sybasec8/server/bin
```

Since the Sybase CEP Server loads these files at startup, if the server is running when the files are copied, the server must be restarted to load these files.

You should thoroughly debug UDFs in a context other than the Sybase CEP environment since the Sybase CEP software contains no debugging information.

You must also update the `c8-services.xml` file to specify the library and function names. See the *Sybase CEP Installation Guide* for details about updating the `c8-services.xml` file.

Sybase CEP Source files

Provides descriptions for Sybase CEP source code files for the RPC plugins.

Sybase CEP provides three source code files related to RPC plugins:

- c8_rpc_http.cpp: This is the Sybase CEP RPC plugin that uses the HTTP protocol.
- c8_rpc_soap.cpp: This is the Sybase CEP RPC plugin that uses the SOAP protocol.

- c8_http_client.h: This header file is used with BOTH the preceding files (the SOAP protocol is built on top of the HTTP protocol, and therefore both the SOAP and HTTP plugins use material in the HTTP header file).

You can use these as they are or you may make a copy and customize them.

Compiling an RPC Plugin

To compile an RPC plugin, you must specify appropriate settings for your compiler.

1. Specify that the compiler should generate a shared object file (.so) or a .DLL file.
2. Ensure the list of "include" directories includes the directory that holds `c8_http_client.h`. This could be either the directory in which Sybase CEP originally supplied this file (`server\sdk\c\examples`) or a separate directory that you have created and to which you have copied the file.

Compiling an RPC Plugin for Windows

This section explains the process of compiling an RPC plugin for a Windows platform.

If you are on Microsoft Windows and are using Microsoft's Visual Studio, please do the following:

1. Start Microsoft Visual Studio.
2. Create a project file by going to the menu and clicking on File -> New -> Project.
   a) Specify that this is a Visual C++ Project.
   b) Click on "Win32".
   c) In the right-hand pane, Click on "Win32 Project".
   d) Fill in the name that you'd like to use for your project.
   e) Browse and specify the directory in that you want the project to be stored.
   f) Click OK.
   g) The next window to appear will be the "Win32 Application Wizard" window. On the left, click on "Application Settings", then click on "DLL".
   h) Click on "Finish".
3. Microsoft Visual Studio creates a simple .cpp file to use as a starting point. We recommend that you remove all the contents of this file and insert the C code for the RPC plugin. Make sure that the source code file includes code similar to the following:

```
#include "c8rpc.h"
    #include "c8_http_client.h"
    // exporting stuff from dll
    #if defined(_MSC_VER)
    #define USER_ADAPTER_EXPORT __declspec( dllexport )
    #else // defined(_MSC_VER)
    #define USER_ADAPTER_EXPORT
    #endif // defined(_MSC_VER)
    // forward declarationsof RPC plugin callback functions
    #ifdef __cplusplus
```

```
    extern "C" {
    #endif /* __cplusplus */
    USER_ADAPTER_EXPORT
    C8Status
c8_rpc_http_initialize
(C8Rpc * rpc_ctx);
    USER_ADAPTER_EXPORT
    C8Status
c8_rpc_http_execute
    (C8Rpc * rpc_ctx,
                           const C8Message *);
    USER_ADAPTER_EXPORT
    void
   c8_rpc_http_shutdown
   (C8Rpc * rpc_ctx);
    #ifdef __cplusplus
    } /* extern "C" */
    #endif /* __cplusplus */
```

You can customize the names shown in bold but the names used in the source code file and the names used in the `c8-services.xml` file must match.

4. If you have other C-language source files that you need, add them to the project.
5. Update several settings that are available in the Property Pages for this project:
   a) Update the list of directories to search for include files.

      To do this, go to the menu, click on "Project" and then on "Properties". You should get a new window titled something like "MySample Property Pages".

      In the left-hand pane of this window, click on "Configuration Properties", then on "C/C++", and then on "General".

      The right-hand pane should now show a list of settings that you may modify.

      Click in the field to the right of Additional Include Directories and add the directory that contains the files `c8rpc.h` and `c8_client_http.h` (which are included with the Sybase CEP product).

      On 32-bit Microsoft Windows, this directory containing c8rpc.h is typically:

      ```
      C:\Program Files\SybaseC8\Server\sdk\c\include
      ```

      On 64-bit Microsoft Windows, this directory is typically:

      ```
      C:\Program Files (x86)\SybaseC8\Server\sdk\c\include
      ```

      The directory containing `c8_http_client.h` depends upon where you've copied this file to.

      You can also add other directories if necessary.
   b) Turn off precompiled headers.

To do this, go to the left-hand pane in the Property Pages window, click on "C/C++" and then on Precompiled headers, then click on "Create/Use Precompiled Header" and set it to "Not Using Precompiled Headers".

c) Add the Sybase CEP library directory to the list of library directories. To do this, go to the left-hand pane of the Property Pages window, click on "Linker" and then on "General".

In the field to the right of "Additional Library Directories", add the directory that contains the Sybase CEP library.

On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\SybaseC8\Server\sdk\c\lib
```

On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\SybaseC8\Server\sdk\c\lib
```

d) Tell the linker not to include debugging information. To do this, go to the left-hand pane of the Property Pages window, click on "Linker" and then on "Debugging". For the field Generate Debugging Info, change the value to "No".

e) Add a dependency on the c8_sdk_server_lib.lib file. To do this, go to the left-hand pane of the Property Pages window, click on "Linker" and then on "Input".

In the field to the right of Additional Dependencies, enter

```
c8_sdk_server_lib.lib nspr4.lib plc4.lib libxml2.lib
```

You do not need to enter the complete path; entering the file name is sufficient.)

If you'd like to double check that you haven't skipped a step, you can look at the Command Line for the C/C++ compiler and for the Linker. These show the command-line parameters passed from Microsoft's GUI IDE to the command-line compiler and linker.

To view the command line for the C/C++ compiler, go to the left-hand pane of the Property Pages window, click on "C/C++" and then click on "Command Line".

Now that you have entered all the project properties, click the "OK" button on the Property Pages window. At this point, you should be ready to compile.

6. To compile, use the appropriate option on the Build menu, for example, "Build MySampleRPC".

7. Copy the DLL file to the server's bin directory. In Microsoft Windows, this directory defaults to:

```
C:\Program Files\SybaseC8\Server\bin
```

Using a Command Line Compiler

This section explains the process of using a command line compiler to compile an RPC plugin.

If you are using a command line compiler (such as "cc"), please perform the following steps.

1. Create a script to compile the RPC plugin. The example below is based on the "cc" compiler found on many UNIX-like operating systems.

```
cc -I${HOME}/sybasec8/server/sdk/c/include \
        -o libmma.so \
        -L${HOME}/sybasec8/server/sdk/c/lib/ \
        -lc8_sdk_server_lib -lnspr4 -lplc4 -llibxml2\
        -fPIC \
        -shared \
        mma.cpp
```

   where `mma.c` is the name of your source code file and `libmma.so` is the name you'd like to use for the library file.

   `-I` specifies the directory(s) to be searched for #include files.

   `-o` specifies the name of the output file. Typically, this is a shared library file.

   `-L` specifies the directory(s) to search for library files that need to be linked with this one.

   `-l` specifies the names of the Sybase CEP libraries to link with to use the Sybase CEP functions.

   `-fPIC` specifies that the compiler should generate Position-Independent Code for dynamic linking.

   `-shared` specifies that the output will be a shared object library file as opposed to a stand-alone executable program.
2. Compile the code by executing the script.
3. Copy the shared library file to the server's bin directory. For example:

   `/home/<userid>/sybasec8/server/bin`

User Authentication

In this section, we discuss using the Sybase CEP C/C++ SDK with user authentication.

We assume that you have already read *User-Defined Functions and Plugins* on page 39 and *Server Plugins* on page 327. If you are not using the User Authentication feature of Sybase CEP Engine Enterprise Edition, you do not need to read this section.

In the first part of this section, we show the functions in the Sybase CEP C/C++ SDK that allow you to get and set credentials (ID and password).

In the second part of this section, we provide information about how to write your own plugin to authenticate users.

User Credentials API

A credential is a data structure that contains a user ID and password.

If you are using the Enterprise Edition of the Sybase CEP Engine and you have enabled the User Authentication feature, you may need to get and set credentials. To perform an action that requires authentication, create a set of credentials, fill them in with the user's ID and password, and pass those credentials to the function that performs the action that you need.

Specifying credentials will add access only to the resources, such as workspaces and streams, and actions, such as create, read, startProgram, for which the specified user was granted privileges in the ACL file. If the user wasn't granted access to a particular resource/action, then specifying Credentials will not give them access.

The following parts of the Sybase CEP C/C++ API. These functions allow you to get and set credential information. The functions that actually use these credentials are documented elsewhere in this manual.

- **C8UserCredentials* C8UserCredentialsCreate (void); –** Purpose: Creates an empty UserCredentials structure. The credentials may be filled in with a user name and password by calling the functions `C8UserCredentialsSetUser()` and `C8UserCredentialsDestroy()`. The credentials must be destroyed with `C8UserCredentialsDestroy()`.

  Parameters:

  - None

  Returns: A pointer to an empty UserCredentials structure. Returns NULL if there is an error.
- **C8Status C8UserCredentialsSetUser (C8UserCredentials* i_credentials, const C8Char* i_user); –** Purpose: Fills in the username field in the user credentials structure.

  Parameters:

  - i_credentials: A pointer to the instance of C8UserCredentials of a the username should be written.
  - i_user: The username.

  Returns: C8_OK on success; C8_FAIL otherwise.
- **C8Status C8UserCredentialsSetPassword (C8UserCredentials* i_credentials, const C8Char* i_password); –** Purpose: Fills in the password field of a user credentials structure.

  Parameters:

  - i_credentials: A valid pointer to user credentials.
  - i_password: The password.

Returns: C8_OK or C8_FAIL.

- **const C8Char * C8UserCredentialsGetUser (const C8UserCredentials* i_credentials); –** Purpose: Returns the username for the given credentials.

  Parameters:

  - i_credentials: The credentials for which you want to look up the user name.

  Returns: Returns the username for the given credentials. The returned pointer is valid as long as the user credentials pointer is valid and must not be deallocated.
- **const C8Char *C8UserCredentialsGetPassword (const C8UserCredentials* i_credentials); –** Purpose: Returns the password for the given credentials.

  Parameters:

  - i_credentials: The credentials for which you want to look up the user's password.

  Returns: Returns the password for given credentials. The returned pointer is valid as long as the user credentials pointer is valid and must not be deallocated.
- **void C8UserCredentialsDestroy (C8UserCredentials* i_credentials); –** Purpose: Destroys user credentials. Any constant pointers (for example, pointers to the user name or password) returned by calling functions with this set of credentials are invalidated upon return from this method.

  Parameters:

  - i_credentials: A pointer to the credentials to destroy.

  Returns: Nothing.

Creating Your Own Authentication Plugin

enables you to write a plugin to authenticate users. This section describes the API for writing your own plugin.

Note: The can use only one authentication plugin at a time.

An authentication plugin uses a plugin pointer and an authentication context. The plugin pointer is passed to the plugin each time the `initialize()`, `authenticate()`, or `shutdown()` function is called. You then pass this pointer to other library functions when calling them. The data structure that the plugin pointer points to is opaque, meaning you do not need to read or change it.

The authentication context holds information about the user, such as their username, ID, and the names of the groups that they are a member of. The authentication content also holds information about the authentication results.

The following is an example of an authentication plugin:

```
/**
 * The plugin pointer
```

```
 */
typedef struct C8AuthPluginImp C8AuthPlugin;
/**
 * Authentication result codes
 */
#define C8_AUTH_SUCCESS 0
#define C8_AUTH_FAILURE 1
#define C8_AUTH_ERROR   2
#define C8_AUTH_UNAVAIL 3
/**
 * The authenticationg context
 */
typedef struct C8AuthContextImp C8AuthContext;
```

The following functions allow you to get and set information in the authentication context variable.

- **const char * C8AuthContextGetUsername (C8AuthContext *ctxt); –** Purpose: Returns the user name from an existing authorization context.

  Parameters:

  - ctxt: A pointer to the authorization context you want to retrieve the user name from.

  Returns: A string containing the username.
- **const char * C8AuthContextGetpassword (C8AuthContext *ctxt); –** Purpose: Returns the user password from an existing authorization context.

  Parameters:

  - ctxt: A pointer to the authorization context you want to retrieve the user password from.

  Returns: A string containing the password.
- **void C8AuthContextSetResult (C8AuthContext *ctxt, C8Int res); –** Purpose: Sets a value that indicates the result of the authentication.

  Parameters:

  - ctxt: A pointer to the authorization context you want to set the result from.
  - res: The result. C8_AUTH_SUCCESS indicates that the user was authenticated.

  Returns: Nothing.
- **void C8AuthContextAddGroup (C8AuthContext *ctxt, const char *groupName); –** Purpose: Adds a group to the list of groups that the user is a member of. A single user may be a member of multiple groups, so you can call this function more than once for a single user. Each time you call, the group name that you specify will be added to the list of group names for user.

  Parameters:

  - ctxt: A pointer to the authorization context for which you want to set the result.

- groupName: The name of a group this user is a member of.

Returns: Nothing.

Note: Add the groups after you have authenticated the user. Only users who have been successfully authenticated should have the list of groups.

As with any plugin, an authentication plugin contains "`initialize()`", "`execute()`" and "`shutdown()`" functions. Below we show some examples with the correct input parameter types and return types. The names for these functions can be different, but you can specify the actual names of the function in the "AccessControl/Authentication/plugin" section of your server configuration file.

- **C8Status c8authplugin_demo_initialize(C8AuthPlugin *plugptr); –** Purpose: Performs any initialization that the plugin needs. This may involve allocating memory, opening a file, or connecting to an authentication servic.

  Parameters:

  - plugPtr: A pointer to a C8AuthPlugin object.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Status c8authplugin_demo_authenticate(C8AuthPlugin *plugPtr, C8AuthContext *); –** Purpose: Authenticates a user.

  Parameters:

  - plugPtr: A pointer to a C8AuthPlugin object.
  - i_credentials: A pointer to the credentials to of the user.

  Returns: C8_OK if successful, C8_FAIL otherwise.
- **C8Status c8authplugin_demo_shutdown(C8AuthPlugin *plugPtr); –** Purpose: Performs any cleanup that the plugin needs. This may involve de-allocating memory, or closing a file.

  Parameters:

  - plugPtr: A pointer to a C8AuthPlugin object.

  Returns: C8_OK if successful; C8_FAIL otherwise.

If you want to preserve information across calls to the execute/authenticate function, you can store information by using the following:

- **void* C8AuthPluginGetSessionState (C8AuthPlugin *plugPtr); –** Purpose: Gets plugin's state. Returns NULL if no state was set. The pointer points to a user-defined structure that holds information that the user needs each time the execute function is called.

  Parameters:

- plugPtr: A pointer to a C8AuthPlugin object.

Returns: C8_OK if successful, C8_FAIL otherwise.

- **void C8AuthPluginSetSessionState (C8AuthPlugin *plugPtr, void *statePtr); –** Purpose: Sets the plugin's state. The pointer points to a user-defined structure that holds information that the user needs each time the execute/authenticate function is called. The plugin is responsible for destroying private data during the "shutdown" callback

  Parameters:

  - plugPtr: A pointer to a C8AuthPlugin object.
  - statePtr: A pointer to the "state" information that the user wants to preserve.

  Returns: C8_OK if successful; C8_FAIL otherwise.

The state information is not retained if the server restarts.

The instructions for compiling a C/C++ plugin are virtually identical to the instructions for compiling an in-process adapter. See *Step-by-Step Instructions for Creating an In-process Adapter* on page 189 for more information. Note that you will need to include the `c8auth_plugin.h` file

Plugin Configuration

This section describes where to customize the authentication plugin in the Sybase CEP Server configuration file, and provides a sample configuration section.

The authentication plugin is declared in the SybaseC8/Security/AccessControl/Authentication/Plugin section of the Sybase CEP Server configuration file named `c8-server.conf`. Below is a sample configuration section. You can customize the actual names of the library file and the three functions, all of which are boldfaced in the listing below.

```
<section name="AccessControl">
  ...
  <section name="Authentication">
     <!-- Sample authentication plugin configuration -->
     <section name="Plugin">
       <preference name="LibraryName"
                   value="
c8authplugin_demo_lib
"/>
       <preference name="InitializeFunction"
                   value="
c8authplugin_demo_initialize
"/>
       <preference name="AuthenticateFunction"
                   value="
c8authplugin_demo_authenticate
"/>
       <preference name="ShutdownFunction"
                   value="
```

```
c8authplugin_demo_shutdown
"/>
     </section>
   </section>
 </section>
```

Custom configuration parameters can be passed to a plugin by specifying a preference with any name in the SybaseC8/Security/AccessControl/Authentication/Plugin section of the server configuration file named `c8-server.conf`. For example, in the htpasswd plugin provided by Sybase CEP, the paths and names of the htpasswd and htgroup files are specified this way. A generic example is shown below:

```
<section name="AccessControl">
    ...
     <section name="Authentication">
         ...
         <section name="Plugin">
             ...
             <preference name="MyPreference"
                         value="MyValue"/>
         </section>
     </section>
 </section>
```

You can then call the following function to read the preference:

```
/**
 * Gets specified preference from server config file.
 * The caller is responsible for de-allocating the returned value.
 *
 * @param pref - name of preference to read
 *               (under SybaseC8/Security/AccessControl/
Authentication/)
 * @param def_val - default value to return if preference is not
present
 * @return - string value of preference (copy of default if not
present).
 */
C8Char* C8AuthPluginGetPreference (C8AuthPlugin *plugPtr,
                                   const C8Char *pref,
                                   const C8Char *def_val);
```

For example, to read the "MyPreference" preference above, one would call:

```
C8Char *pref = C8AuthPluginGetPreference(plugPtr, "MyPreference",
                                                  "default");
```

Library-Wide Initialization and Shutdown

This section describes how to write functions to perform library intitalization and shutdown.

One library (.dll or .so file) for Sybase CEP Server may contain more than one UDF, more than one in-process adapter, or more than one server plugin. In some cases, you may want to have `initialize()` and `shutdown()` functions that apply to the entire library in addition to those that apply for a particular in-process adapter or plugin. These functions are called once when that the library is loaded or unloaded by Sybase CEP Server.

To implement this, write two functions with the following signatures:

- **C8Int c8_library_initialize() –** Purpose: This function is called immediately after the library is loaded. It can do any initialization required for the library as a whole, and also return a value to the Sybase CEP system indicating whether the library is unloadable or not.

  Parameters: None.

  Returns: Any one of the values in the C8LibraryResult enumeration:

  - C8_LIBRARY_INIT_ERROR: Initialize failed. The library will not be loaded. The `c8_library_shutdown()` function will be called if it is present.
  - C8_LIBRARY_INIT_OK: Initialize completed successfully.
  - C8_LIBRARY_INIT_OK_CAN_UNLOAD: The initialization completed successfully, and Sybase CEP Server may unload the library if there are no remaining references to it. For example, if the library is a UDF library, and if at some future time the server has unloaded all projects that call UDFs in this library, then it would be safe to unload this library.
- **void c8_library_shutdown() –** Purpose: This function is called immediately prior to the library being unloaded. It does any cleanup required for the library as a whole.

  Returns: Nothing.

These functions are purely optional. In your library, you can include either function, both functions, or neither function.

The function prototypes and the C8LibraryResult enumerations which defines the return codes such as C8_LIBRARY_INIT_ERROR are in the `c8server.h` file. This file is located in the "include" subdirectory of the Sybase CEP C/C++ SDK directory. If you installed to the default directory on Microsoft Windows, this will be:

```
C:\Program Files\SybaseC8\Server\sdk\c\include
```

If you installed to the default directory on a UNIX-like operating system, this will be:

```
/home/user/sybasec8/server/sdk/c/include
```

Sybase CEP Java SDK

Use the Sybase CEP Java SDK to write out-of-process adapters, control Sybase CEP Engine, register queries, and perform a variety of other tasks.

This chapter shows you how to examine, compile, and run the examples that ship with the product, which perform some of the most common tasks involved in writing an adapter.

Warning! Sybase CEP only validates against the Sun JDK, and not any other Java development environment. See " Sybase CEP Engine Third-Party Software Dependencies" in the *Sybase CEP Installation Guide* for version information.

Locating Java files

When you install Sybase CEP Server or Sybase CEP Studio, you also install the Sybase CEP Java 1.5 SDK.

For installation instructions, see the *Sybase CEP Installation Guide*. All of the Java 1.5 SDK files are located in the directory java5 under the Sybase CEP Server or Sybase CEP Studio installation directory. The SDK reference documentation is in the directory java5/doc. Open the file index.html in a Web browser to get started with the reference documentation.

Sybase CEP ships several examples with the Java SDK, which you can find under `java5/examples/com/sybasec8/sdk/examples`. The examples are annotated to help you understand the purpose of each line of code.

Setting Up Your Environment

Describes how to modify the CLASSPATH variable in order to set up your environment.

To set up your development environment for the Sybase CEP Java SDK is, modify your CLASSPATH variable to include `java5/c8-sdk-java5.jar` under your installation directory. A manifest file in this JAR file automatically includes the files under java5/lib. To use a different version of any of the files under java5/lib, specify each of them separately in your CLASSPATH.

Using the Java Examples

TheJava 1.5 SDK examples provide sample code that performs all of the most common tasks you need to perform while writing an adapter with the SDK.

The examples progress from simple, basic tasks to more complicated implementations. Your best path for learning the Java 1.5SDK is to trace the source code of each example before compiling and executing it, using the HTML files under java5/doc for syntax and usage reference.

As you examine and run the Java examples, read the embedded comments for additional information. A description of each example generally begins with the comments in the **runExample** method and frequently begins with the words "The core of the example starts here."

Most of the examples use the same internal CCL query module mimicking temperature readings from sensors at different locations.

Every example, with one exception, requires you to identify the location of the Sybase CEP compiler (bin/c8_compiler under your Sybase CEP Studio or Sybase CEP Server installation) by passing the –D option to the JVM to specify a value for the variable c8.compilerPath.

JavaAPI dependencies files in Maven

Sybase CEP provides a set of Maven setup scripts which enable you to build Sybase CEP Java SDK example files.

Sybase CEP also provides a `pom.xml` file which contains the dependencies required to build the example files. The example files enable you to link Sybase CEP API jar files into their systems and facilitate easier Java code compilation.

The Maven setup scripts—setup_maven.bat and setup_maven.sh— create the proper directory and repository structures and copy the files from the local file system to the appropriate locations on your systems.

Examining Example 1: Subscribing to a Stream

This section provides an example of subscribing to a stream, which is necessary for writing an output adapter.

Subscribing to a stream

To write an output adapter, you must subscribe to a stream.

Typically, you read the data being fed to a stream by a query module, process that data according to your application, and then send it to its final destination. For more information about streams, see *Data Stream* on page 8. For more information about adapters, see *Adapters* on page 13.

The file `Example_01_GettingDataFromAStream.java` contains source code demonstrating the basic steps you perform to subscribe to a data stream:

1. Create an instance of a client.
2. Create a CCL URI identifying the stream.
3. Subscribe to the stream.
4. Read data from the stream and process it.
5. Disconnect from the stream.

Creating an Engine Client in Java

This section describes how to create an engine client in Java in Sybase CEP.

First you create an instance of a client to connect to and communicate with an instance of Sybase CEP Server, as part of the constructor:

```
engineClient = SDK.getEngineClientFactory().newEngineClient(
   serverUrl, CredentialsFactory.newCredentials(userName,
   password));
```

This line creates a new engine client to run operations on a particular Sybase CEP Server process. The **newEngineClient** method takes parameters specifying the HTTP URL of the host running Sybase CEP Server (such as **http://mymachine.mycompany.com:6789**) and the authentication credentials to use when connecting with a server that requires authentication. The example defaults to a server process on the local host with no authentication required. When you run any of the Java 1.5 examples, all of which contain this code, you can pass a server URL, account name, and password on the command line to override the default behavior, if needed for your environment.

Creating a URI

Create CCL URIs for streams you want to retrieve data from.

Create the CCL URI:

```
String cclUri = CclUriFactory.newCclStreamUri(
   serverUrl, C8_JAVA_SDK_EXAMPLES_WORKSPACE,
   "SensorNetwork", "SensorReadings");
```

The parameters passed to this method include the URL of the Sybase CEP Server process, the name of the workspace containing the module, the name of the module containing the stream, and the name of the stream. In your application, this stream information might be hard-coded, read from an external configuration file, or be from a stream belonging to a query created internally to your application. The example does the latter, calling prepareExampleQuery to create a simple stream containing generated random data.

Subscribing to a stream in Java

A code example for subscribing streams in Java.

To subscribe to a stream:

```
subscription = engineClient.subscribeToStream(cclUri);
```

The code is using a method of the engine client that was created in the first step and passing it the CCL URI generated in the second step.

Reading data from a stream in Java

This section provides a code example for reading tuple rows from streams in Java.

Now you can read a tuple (row) from the stream:

```
Tuple tuple = subscription.getNextTuple(1000);
```

The parameter specifies a timeout for this blocking call, in milliseconds.

In your application you process the data in the tuple according to your design, but this example simply prints out the tuple using the message serializer.

Make sure to check the return value of getNextTuple for null, which indicates that no row is available.

Disconnecting from a stream in Java

At the end of processing, the example disconnects the subscription.

```
subscription.disconnect();
```

Compiling and Running in Java

This section describes the steps and commands you need to compile and run projects in Java.

Navigate to the examples directory under the installation: sdk/java5/examples.

Enter the following command:

```
javac com/sybasec8/sdk/examples/
Example_01_GettingDataFromAStream.java
```

Run the compiled project by entering the following command, replacing *<PathToC8Compiler>* with the full path to and name of the Sybase CEP compiler on your system, typically **bin/c8_compiler** under the installation directory:

```
java –Dc8.compilerPath=
<PathToC8Compiler>

 com.sybase.c8.sdk.examples.Example_01_GettingDataFromAStream
```

Running the example produces output that looks similar to this:

```
Ts,SensorID,Reading
1206999928816643,Rw,2.08615
1206999928926019,YH,9.3957
1206999929035394,Xo,6.69305
1206999929144770,qj,5.507
1206999929254146,N4,2.71035
1206999929363521,UF,4.9583
1206999929472897,rk,0.62637
1206999929582273,eV,4.72496
1206999929691649,TY,3.39727
1206999929801024,Mn,8.46074
1206999929910400,pY,1.80193
1206999930019776,OR,8.98612
1206999930129151,by,6.31523
```

The first line lists the column names from the schema, while the remaining output is a list of the rows from the data stream.

Other Examples

This section describes each of the other Java 1.5 examples, briefly explaining the purpose, listing the classes and methods of primary interest for the particular example, and specifying the location of relevant information in the documentation.

Publishing to a stream in Java

This section provides an example that demonstrates how to publish to a stream in Java.

Name	Example_02_PublishingDataToAStream

Description	This example demonstrates the steps to take data from an outside source and publish it to a stream feeding a query module. necessary to publish data to a stream. This is a typical activity when writing an input adapter.
Notable Classes and Methods	**Publisher** engineClient.**createPublisher**(cclUriOfInput-Stream) publisher.**publishRow**(point) publisher.**disconnect**()
Related Information	For more information about streams, see *Data Stream* on page 8. For more information about adapters, see *Adapters* on page 13.

Controlling the engine in Java

This section contains an example that demonstrates how to perform engine control.

Name	Example_03_ControllingTheSybaseCEPEngine
Description	This example presents methods used to perform engine control: retrieving general server, workspace, and project information; stopping a project; creating a workspace; and destroying a workspace.
Notable Classes and Methods	engineClient.**getServerVersion**() SDK.**getVersion**() SDK.**isCompatibleWithServerVersion**(server-Version) **WorkspaceInfo** engineClient.**getWorkspaces**() ws.**getName**() ws.**getDescription**() engineClient.**getProjectsInWorkspace** engineClient.**stopProject** engineClient.**createWorkspace** engineClient.**destroyWorkspace**
Related Information	For more information about engine control, see *Engine Control: Overview* on page 21.

Registering a Query in Java

An example that demonstrates how to register a query in java.

Name	Example_04_RegisteringAQuery
Description	This example demonstrates how to register a query. This same code is used in examples 1 and 2 to create a stream for subscribing and publishing. In this example, the primary purpose is to register a query, so the example produces no output.
Notable Classes and Methods	**RegisterableQueryFactory.newInstance**() rqf.**newQuery** engineClient.**registerQuery**(query)
Related Information	For more information about registering a query, see *Dynamically Registering Queries and Streams* on page 26.

Compiling and starting a project in Java

This section contains an example that demonstrates how to compile and start a project in Java.

Name	Example_05_CompilingAndStartingAProject
Description	This example demonstrates how to compile and start a project. The CCL and project files for this example are under **java5/examples/projects**. If you do not run this example from **java5/examples**, you need to set the variable **c8.examples.example05ProjectDir** with the D option to the path to the projects directory. Note that this example includes intentional errors to demonstrate compiler messages.
Notable Classes and Methods	CclCompilerFactory.newInstance().**newLocalCompiler**() localCompiler.**compile** CclCompilerFactory.newInstance().**newRemoteCompiler** e.**getCompilerOutput**()
Related Information	For more information about compiling and starting projects, see *Commands* on page 23.

Exploring Value Types in Java

An example that demonstrates how to create and modify value types in Java.

Name	Example_06_ExploringSybaseCEPValueTypes
Description	This example creates and manipulates objects of various types, demonstrating the data types available in the Java 1.5SDK. You do not need to set the compiler variable to run this example since it does not register or compile any queries.
Notable Classes and Methods	**ValueFactory** vf = ValueFactory.newInstance()

Examining Schemas in Java

An example that demonstrates how to work with schemas in Java.

Name	Example_07_ExploringStreamSchema
Description	This example demonstrates how to read the schema of a stream or row, create a schema, and compare two schemas.
Notable Classes and Methods	**Schema** engineClient.**getStreamSchema**(cclUri) t.**getSchema()** **SchemaFactory** sf.**newSchema** **SchemaColumn** schema.**toCclDefinition** schema1.**equivalentTo**
Related Information	For more information about schemas, see *Schema* on page 8.

Working with Tuples in Java

An example that demonstrates how to create and modify tuples in Java.

Name	Example_08_ExploringTuples
Description	This example presents methods for creating and manipulating tuples (data rows).

Notable Classes and Methods	**Tuple** tuple.**getTimestamp**() tuple.**getValues**() tuple.**getValuesWithTimestamp** tuple.**getValuesAsStrings**() tuple.**getValuesAsStringsWithTimestamp** MessageFactory mf.**newTuple**
Related Information	For more information about tuples, see *Data Streams and Messages* on page 7.

Retrieving the server status in Java

An example that demonstrates how to retrieve server status information in Java.

Name	Example_09_QueryingSybaseCEPEngineStatus
Description	This example demonstrates how to retrieve information about an instance of Sybase CEP Server and its activities.
Notable Classes and Methods	**StatusInfo** engineClient.**getManagerStatus**() managerStatus.**getObjectName** managerStatus.**getValue** managerStatus.**getObjectCount** workspaceStatus.**getMessageCount** workspaceStatus.**getMessageName** workspaceStatus.**getValue**
Related Information	For more information about status, see *Data Streams and Messages* on page 7 and *Status Information* on page 519.

Publishing Asynchronously

An example that demonstrates how to publish data asynchronously.

Name	Example_10_PublishingAsynchronously

Description	This example demonstrates how to publish data to a stream asynchronously. This is a variation of Example 2 that produces very similar output, since the difference is not in the data but in how the data is published. In particular, this example shows how to define and register a listener. Note that the actual publish method is the same for both synchronous and asynchronous publishing.
Notable Classes and Methods	publisher.setOption Publisher.Listener publisher.setListener

Subscribing Asynchronously

An example that demonstrates how to subscribe to a stream asynchronously.

Name	Example_11_SubscribingAsynchronously
Description	This example demonstrates how to subscribe to a stream and retrieve data asynchronously. This is a variation of Example 1 that produces very similar output, since the difference is not in the data but in how the data is retrieved. In particular, this example shows how to define and register a listener.
Notable Classes and Methods	subscription.setOption SampleListener subscription.s()

Working with Bundles in Java

An example that demonstrates how to work with bundles in Java.

Name	Example_12_WorkingWithMessageBundles
Description	This example illustrates how to create, retrieve, and process messages as part of a bundle.
Notable Classes and Methods	**MessageBundle** mf.**newMessageBundle** bundle.**getMessages**() if (msg instanceof Tuple) Subscription.FLATTEN_BUNDLES subscription.**getNextMessage** bundle.**getSize**()

Related Information	For more information about bundles, see *Bundles* on page 11.

Guaranteeing Message Delivery in Java

An example that demonstrates how to use guarenteed delivery in Java.

Name	Example_13_WorkingWithGuaranteedDelivery
Description	This example illustrates how to publish and subscribe with guaranteed delivery.
Notable Classes and Methods	engineClient.**createPublisherWithGuaranteedDelivery** publisher.**getLastBatchId**() **BatchOfMessages** mf.**newBatchOfMessages** subscription1.**getNextBatchOfMessages** receivedBatch1.**getTuples**() engineClient.**resumeSubscriptionWithGuaranteedDelivery** receivedBatch1.**getBatchId**()
Related Information	For more information about guaranteed delivery, see *Implementing Guaranteed Processing* on page 89.

Registering a Query with parameters in Java

An example that demonstrates how to register a query with parameters in Java.

Name	Example_14_RegisteringParameterizedQuery
Description	This example illustrates how to register a query with parameters.
Notable Classes and Methods	**Parameter** parameterFactory.**newParameter**
Related Information	For more information about parameters, see *Engine Control: Overview* on page 21.

Working with URIs in Java

An example that demonstrates how to create CCL URIs in Java.

Name	Example_15_WorkingWithStreamURIs

Description	This example demonstrates how to create a new CCL URI and then convert the URI to an HTTP URL.
Notable Classes and Methods	engineClient.resolveUri
Related Information	For more information about URIs, see *Stream URIs* on page 365.

Querying a public window in Java

An example that demonstrates how to query a public window in Java.

Name	Example_16_QueryingWindowState
Description	This example demonstrates how to query a public window.
Notable Classes and Methods	engineClient.**getWindowState** WindowQueryFactory.newInstance().**newSQLQuery**
Related Information	For more information about public windows, see *Public Windows* on page 65.

Working with Parallel Queries in Java

An example that demonstrates how to work with parallel queries in Java.

Name	Example_17_WorkingWithParalellizedQueries
Description	This example demonstrates how to work with parallel queries.
Notable Classes and Methods	engineClient.**resolveClusterUri**
Related Information	For more information about parallel queries, see the Sybase CEP *Installation Guide* .

Sybase CEP .NET3 SDK

Use the Sybase CEP .NET3 SDK to write out-of-process adapters, control Sybase CEP Engine, register queries, and perform a variety of other tasks. This chapter shows you how to examine, compile, and run the examples that ship with the product, which perform some of the most common tasks involved in writing an adapter.

Locating .NET files

When you install Sybase CEP Server or Sybase CEP Studio, you also install the Sybase CEP .NET3 SDK.

For installation instructions, see the *Sybase CEP Installation Guide* . You can find all of the .NET3 SDK files in the directory **net3** under the Sybase CEP Server or Sybase CEP Studio installation directory. The SDK reference documentation is in the directory **net3/doc**. Open the file `Documentation.chm` to get started with the reference documentation.

Sybase CEP ships several examples with the .NET3 SDK. You will find compiled versions of the examples under **net3/examples** and the corresponding source code under **net3/examples/src**. The examples are annotated to help you understand the purpose of each line of code.

Using the .NET3 examples

The .NET3 SDK examples provide sample code that performs all of the most common tasks you will perform while writing an adapter with the SDK.

The examples progress from simple, basic tasks to more complicated implementations. Your best path for learning the .NET3 SDK is to trace the source code of each example before executing it, using the reference documentation (`net3/doc/Documentation.chm`) for syntax and usage reference.

As you examine and run the examples, read the embedded comments for additional information. A description of each example generally begins with the comments in the **RunExample** routine and frequently begins with the words "The core of the example starts here."

Most of the examples use the same internal CCL query module mimicking temperature readings from sensors at different locations.

In order to run the examples, copy the three .dll files from **net3** to **net3/examples**. Every example, with one exception, requires you to identify the location of the Sybase CEP compiler

by setting a value for the variables. This compiler is located in **bin/c8_compiler** under your Sybase CEP Studio or Sybase CEP Server installation.

Examining Example 1

This section contains an example using .NET3 SDK.

Subscribing to a Stream

This section contains an example that demonstrates how to subscribe to a stream.

If you are writing an output adapter, you need to subscribe to a stream. Typically, you read the data being fed to a stream by a query module, process that data according to your application, and then send it to its final destination. For more information about streams, see *Data Stream* on page 8. For more information about adapters, see *Adapters* on page 13.

The file `Example_01_GettingDataFromAStream.cs` contains source code demonstrating the basic steps you perform to subscribe to a data stream:

1. Create an instance of a client.
2. Create a CCL URI identifying the stream.
3. Subscribe to the stream.
4. Read data from the stream and process it.
5. Disconnect from the stream.

Creating an Engine Client in .NET

This section provides a code example for creating an engine client in .NET.

First, create an instance of a client to connect to and communicate with an instance of Sybase CEP Server, as part of the constructor:

```
engineClient = SDK.EngineClientFactory.NewEngineClient(
   serverUrl, CredentialsFactory.NewCredentials(
   userName, password));
```

This line creates a new engine client to run operations on a particular Sybase CEP Server process. The **NewEngineClient** method takes parameters specifying the HTTP URL of the host running Sybase CEP Server (such as **http://mymachine.mycompany.com:6789**) and the authentication credentials to use when connecting with a server that requires authentication. The example defaults to a server process on the local host with no authentication required. When you run any of the .NET3 examples, all of which contain this code, you can pass a server URL, account name, and password on the command line to override the default behavior, if needed for your environment.

Creating a CCL URI

This section provides a code example for creating a CCL URI.

To create the CCL URI of the stream you will be getting data from (subscribing to):

```
string cclUri = CclUriFactory.NewCclStreamUri(
   serverUrl, C8_NET_SDK_EXAMPLES_WORKSPACE,
   "SensorNetwork", "SensorReadings");
```

The parameters passed to this method include the URL of the Sybase CEP Server process, the name of the workspace containing the module, the name of the module containing the stream, and the name of the stream. In your application, this stream information might be hard-coded, read from an external configuration file, or be from a stream belonging to a query created internally to your application. The example does the latter, calling **PrepareExampleQuery** to create a simple stream containing generated random data.

Subscribing to a stream in .NET

This section provides a code example for subscribing to a stream in .NET.

To subscribe to the stream:

```
subscription = engineClient.SubscribeToStream(cclUri);
```

The code uses a method of the engine client that was created in the first step, passing it the CCL URI generated in the second step.

Reading data from a stream in .NET

This section provides a code example for reading tuple data from a stream in .NET.

Read a tuple from the stream. The example calls this statement inside a loop:

```
ITuple tuple = subscription.GetNextTuple(1000);
```

The parameter specifies a timeout for this blocking call, in milliseconds.

In your application,you process the data in the tuple according to your design, but this example simply prints out the tuple using the message serializer.

Make sure to check the return value of **getNextTuple** for NULL, which indicates that no row is available.

Disconnecting from a stream in .NET

This section provides a code example for disconnecting from a stream in .NET.

At the end of processing, the example disconnects the subscription:

```
subscription.Disconnect();
```

Compiling and Running in .NET3

This section describes how to compile and run example source code in .NET3.

You can compile the example source code from the command line using the csc compiler and referencing the file `C8-x.y.z.dll`, where x.y,z is the Sybase CEP Engine version number. You can also load the project file `Examples.csproj` into Visual Studio, set up a reference to the file `C=8.x.y.z.dll`, and then compile from within Visual Studio.

Running the first example produces output that looks similar to this:

```
Ts,SensorID,Reading
1206999928816643,Rw,2.08615
1206999928926019,YH,9.3957
1206999929035394,Xo,6.69305
1206999929144770,qj,5.507
1206999929254146,N4,2.71035
1206999929363521,UF,4.9583
1206999929472897,rk,0.62637
1206999929582273,eV,4.72496
1206999929691649,TY,3.39727
1206999929801024,Mn,8.46074
1206999929910400,pY,1.80193
1206999930019776,OR,8.98612
1206999930129151,by,6.31523
```

The first line lists the column names from the schema, while the remaining output is a list of the rows (tuples) from the data stream.

Other Examples

This chapter describes each of the other .NET3 examples, briefly explaining the purpose, listing the classes and methods of primary interest for the particular example, and specifying the location of relevant information in the documentation.

Publishing to a Stream in

This section contains an example that demonstrates how to publish data to a stream in.NET3 .

Name	Example_02_PublishingDataToAStream
Description	This example demonstrates the steps necessary to publish take data from an outside source and publish it to a stream feeding a query module. This is a typical activity when writing an input adapter.

Notable Classes and Methods	**IPublisher** engineClient.CreatePublisher(cclUriOfInput-Stream) publisher.**PublishRow**(point) publisher.**Disconnect**()
Related Information	For more information about streams, see *Data Stream* on page 8. For more information about adapters, see *Adapters* on page 13.

Controlling the Sybase CEP Engine in .NET3

An example that demonstrates how to perform engine control in .NET3.

Name	Example_03_ControllingTheSybaseCEPEngine
Description	This example presents methods used to perform engine control: retrieving general server, workspace, and project information; stopping a project; creating a workspace; and destroying a workspace.
Notable Classes and Methods	engineClient.ServerVersion SDK.Version SDK.IsCompatibleWithServerVersion(server-Version) IWorkspaceInfo engineClient.GetWorkspaces ws.Name ws.Description engineClient.GetProjectsInWorkspace engineClient.StopProject engineClient.CreateWorkspace engineClient.DestroyWorkspace
Related Information	For more information about engine control, see *Engine Control: Overview* on page 21.

Registering a query

An example that demonstrates how to register a query.

Name	Example_04_RegisteringAQuery
Description	This example demonstrates how to register a query. This same code is used in examples 1 and 2 to create a stream for subscribing and publishing. In this example, the primary purpose is to register a query, so the example produces no output.
Notable Classes and Methods	**RegisterableQueryFactory.NewInstance**() rqf.**NewQuery** engineClient.**RegisterQuery**(query)
Related Information	For more information about registering a query, see *Dynamically Registering Queries and Streams* on page 26.

Compiling and starting a project in C/C++

This section contains an example that demonstrates how to compile and start a project in C/C++.

Name	Example_05_CompilingAndStartingAProject
Description	This example demonstrates how to compile and start a project. The CCL and project files for this example are under net3/examples/projects. If you do not run this example from net3/examples, you need to set the variable c8.examples.example05ProjectDir to the path to the projects directory. Note that this example includes intentional errors to demonstrate compiler messages.
Notable Classes and Methods	CclCompilerFactory.NewInstance().NewLocalCompiler() localCompiler.Compile CclCompilerFactory.NewInstance().NewRemoteCompiler e.CompilerOutput()
Related Information	For more information about compiling and starting projects, see *Commands* on page 23.

Exploring Value Types in .NET

This section contains an example that demonstrates how to create and modify data types in .NET.

Name	Example_06_ExploringSybase CEPValueTypes
Description	This example creates and manipulates objects of various types, demonstrating the data types available in the .NET3 SDK. You do not need to set the compiler variable to run this example since it does not register or compile any queries.
Notable Classes and Methods	ValueFactory vf = ValueFactory.NewInstance()

Examining Schemas in .NET

This section contains an example that demonstrates how to work with schemas in .NET.

Name	Example_07_ExploringStreamSchema
Description	This example demonstrates how to read the schema of a stream or row, create a schema, and compare two schemas.
Notable Classes and Methods	ISchema engineClient. GetStreamSchema(cclUri) t.Schema() SchemaFactory sf.NewSchema ISchemaColumn schema.ToCclDefinition schema1.EquivalentTo
Related Information	For more information about schemas, see *Schema* on page 8.

Working with Tuples in .NET

This section contains an example that demonstrates how to create and modify tuples in .NET.

Name	Example_08_ExploringTuples
Description	This example presents methods for creating and manipulating tuples (data rows).

Notable Classes and Methods	ITuple tuple.Timestamp tuple.Values tuple.GetValuesWithTimestamp tuple.GetValuesAsStrings() tuple.GetValuesAsStringsWithTimestamp mf.NewTuple
Related Information	For more information about tuples, see *Data Streams and Messages* on page 7.

Retrieving Server Status in .NET3

This section contains an example that demonstrates how to retrieve status information in .NET3.

Name	Example_09_QueryingSybaseCEPEngineStatus
Description	This example demonstrates how to retrieve information about an instance of Sybase CEP Server and its activities.
Notable Classes and Methods	IStatusInfo engineClient.GetManagerStatus() managerStatus.GetObjectName managerStatus.GetValue managerStatus.GetObjectCount workspaceStatus.GetMessageCount workspaceStatus.GetMessageName workspaceStatus.GetValue
Related Information	For more information about status, see *Data Streams and Messages* on page 7 and *Status Information* on page 519.

Working with Bundles in .NET

This section contains an example that demonstrates how to work with bundles in .NET.

Name	Example_12_WorkingWithMessageBundles

Description	This example illustrates how to create, retrieve, and process messages as part of a bundle.
Notable Classes and Methods	IMessageBundle mf.NewMessageBundle bundle.Messages if (msg is ITuple) subscription.SetOption(EngineConstants.FLATTEN_BUNDLES, "false") subscription.GetNextMessage bundle.Size
Related Information	For more information about bundles, see *Bundles* on page 11.

Guaranteeing Message Delivery in .NET

This section contains an example that demonstrates how to use guarenteed delivery in .NET.

Name	Example_13_WorkingWithGuaranteedDelivery
Description	This example illustrates how to publish and subscribe with guaranteed delivery.
Notable Classes and Methods	engineClient.CreatePublisherWithGuaranteedDelivery publisher.LastBatchId IBatchOfMessages mf.NewBatchOfMessages subscription1.GetNextBatchOfMessages receivedBatch1.Tuples engineClient.ResumeSubscriptionWithGuaranteedDelivery receivedBatch1.BatchId()
Related Information	For more information about guaranteed delivery, see *Implementing Guaranteed Processing* on page 89.

Registering a Query with parameters in .NET3

An example that demonstrates how to register a query with parameters in .NET3.

Name	Example_14_RegisteringParameterizedQuery
Description	This example illustrates how to register a query with parameters.
Notable Classes and Methods	**IParameter** parameterFactory.**NewParameter**
Related Information	For more information about parameters, see Engine Control: Overview.

Working with URIs in .NET

An example that demonstrates how to create a CCL URI in .NET.

Name	Example_15_WorkingWithStreamURIs
Description	This example demonstrates how to create a new CCL URI (which has also been shown in most of the other example programs) and then convert the URI to an HTTP URL.
Notable Classes and Methods	engineClient.**ResolveUri**
Related Information	For more information about URIs, see Stream URIs.

Querying a Public Window in .NET3

This section contains an example that demonstrates how to query a public window in .NET3.

Name	Example_16_QueryingWindowState
Description	This example demonstrates how to query a public window.
Notable Classes and Methods	engineClient.getWindowState WindowQueryFactory.NewInstance().NewSQLQuery
Related Information	For more information about public windows, see *Public Windows* on page 65.

Working with Parallel Queries in .NET

This section contains an example that demonstrates how to run parallel queries in .NET.

Name	Example_17_WorkingWithParalellizedQueries
Description	This example demonstrates how to work with parallel queries.
Notable Classes and Methods	compilerOptions.Add("NumberOfInstances", "3"; engineClient.ResolveClusterUri
Related Information	For more information about parallel queries, see the *Sybase CEP Installation Guide* .

Sybase CEP Perl SDK

This chapter describes how to use the Sybase CEP Perl SDK, which allows you to write adapters in Perl.

Perl prerequisites

This section describes the software version and modules required to operate Perl in Sybase CEP.

This chapter assumes that you have already installed Perl and are fluent in the language.

Version of Perl: The Sybase CEP Perl SDK has been tested with version 5.8.6 of Perl. Later versions should work. Earlier versions may also work, but have not been tested.

Perl Modules: Sybase CEP Perl adapters depend on the following CPAN (Comprehensive Perl Archive Network) modules:

- URI::URL
- LWP::UserAgent
- Text::CSV
- SOAP::Lite
- XML::DOM
- XML::XPath
- Error

For instructions on installing the Sybase CEP Perl SDK, see *Installation and Configuration* on page 308.

Perl API Interface

This section describes the typical behaviour of adapters and their interaction with the Perl API interface.

A typical adapter performs the following tasks:

1. Acquiring an a URI that will uniquely identify a specific stream and telling the communication layer provided by Sybase CEP how to find the stream.
2. Opening a connection to the stream.
3. Writing the desired data. Typically the write operation is in a loop; the adapter will keep sending multiple rows of data.
4. Closing the connection.

When using the Perl SDK, the URI of the data stream is typically passed to the Perl program as a command-line parameter. To find the URI of an active stream, look at the stream properties page in Sybase CEP Studio. Right-click on the stream, then choose "properties", and read the stream's URI.

When using the Perl SDK, there is no explicit call to close the connection. The connection is closed when you destroy the C8::Subscriber or C8::Publisher object that has the connection.

Perl API

This section describes the three parts of the Perl API: the tuple, publisher, and subscriber APIs.

The Perl API consists of three parts:

- A tuple API for manipulating Sybase CEP messages (C8::Tuple module).
- A publisher API for sending data to Sybase CEP Server (C8::Publisher module).
- A subscriber API for receiving data from Sybase CEP Server (C8::Subscriber module).

With this combination, you can write input or output adapters. For example, in an input adapter you would read data from the data source, transform it into an appropriate format, and then use the publisher API in C8::Publisher to send the data to the server. Similarly, in an output adapter you would use the C8::Subscriber module to read data from the server, then transform the data into the external data format and write the data to its destination.

The complete Sybase CEP Perl SDK, including example source code, is available in the sdk/perl subfolder of your Sybase CEP Server installation.

C8::Tuple

A C8::Tuple object represents a tuple, which is a row of data.

If you are writing data to a stream you create a tuple object, fill it with data, then write it to the stream. Similarly, if you are reading data from a stream, you receive a row of data, read the individual values from the tuple object, and then write the data to your destination.

- **new() –** Creates a new C8::Tuple object.
- **column_names() –** Gets an array that contains the names of the tuple columns.
- **column_names(@val) –** Sets the tuple column names to the values listed in the array.
- **column_name_pos($name) –** Gets the position of a column name. The first column is column 0 (not 1).
- **timestamp() –** Gets the tuple's timestamp, expressed as the number of seconds (not microseconds) since midnight January 1, 1970.

Note: The Perl library contains functions to convert between seconds and more readable formats such as "YYYY-MM-DD HH24:MI:SS".

- **timestamp($val) –** Sets the tuple's timestamp to the specified value, expressed as the number of seconds (not microseconds) since midnight January 1, 1970.

  Note: The Perl library contains functions to convert between seconds and more readable formats such as "YYYY-MM-DD HH24:MI:SS".
- **fields() –** Gets the tuple fields list. This gets a complete row. The data is returned as an array, with one field per array element.
- **fields($val) –** Sets the tuple fields list. This sets a complete row. The data is passed as an array, with one field per array element.
- **field($name) –** Gets a single value from the current tuple. Given the name of the field, returns the value as a string.
- **field($name,$val) –** Sets a single value in the current tuple. The input parameters are the name of the field and the value of the field.
- **as_csv_string() –** Gets the tuple fields value from a CSV string. Note that the first field in the CSV string is the row timestamp value.
- **as_csv_string($val) –** Sets the tuple fields value from a CSV string. Note that the first field in the CSV string is the row timestamp value.

C8::Publisher

Create a C8::Publisher object if you want to write data to a stream. You need one C8::Publisher object per stream.

The following functions allow you to create C8::Publisher objects and connect to streams.

- **new() –** Creates a new C8::Publisher object.
- **connection($uri) –** Creates a publisher connection to the given $uri that identifies a Sybase CEP Stream. Returns 1 if the connection is ready to receive data. Returns an undefined value if an error occurs.
- **write_tuple($tuple) –** Writes a C8::Tuple object to the stream.

Note: There is no explicit call to close the connection. The connection will be closed when you destroy the C8::Publisher object.

C8::Subscriber

Create a C8::Subscriber object if you want to read data from a stream. You need one C8::Subscriber object per stream.

The following functions allow you to create and modify C8::Subscriber objects.

- **new() –** Creates a new C8::Subscriber object.

  connect($uri)

  Creates a subscription connection to the given $uri that identifies a Sybase CEP Stream.

  Returns 1 if the subscription is ready to receive data; returns an undefined value if an error occurs.

read_tuple()

Retrieves the next message from the stream. The first row returned is the title row that contains the names of the fields. This is true even if you connect to a stream that transmits messages to other subscribers. A stream may have more than one subscriber.

Returns the C8::Tuple object; returns an undefined value if there are no more messages in the stream or a connection error occurs.

Note: Calls to `read_tuple()` are blocking calls. The call will not return until a message is received on the stream.

column_names()

Retrieves the list of column names. This is valid only after a successful connect call.

Note: There is no explicit call to close the connection. The connection will be closed when you destroy the C8::Subscriber object.

Perl Input Adapter (Sending Data to a Sybase CEP Stream)

This example demonstrates a simple input adapter that publishes generated messages to the Sybase CEP Stream.

The Perl program connects to the Sybase CEP stream by using a URI specified in the command line. The stream schema is assumed to have exactly two fields ('Symbol' and 'Price') and the generated messages look like the following:

Symbol1, 1

Symbol2, 2

Symbol3, 3

...

The full listing with line numbers is below:

```
1  use C8::Publisher;
2  use C8::Tuple;
3
4  # Get command-line argument(s).
5  my $usage = "$0 <uri>";  # Usage msg to display if err
6  my $uri = shift or
 die "ERROR: URI is missing.\nUsage: $usage\n";
7
8  # Create a publisher object.
9  my $publisher = C8::Publisher->new();
10 $publisher->connect($uri) or
 die "ERROR: cannot publish to '$uri'\n";
11
12 # Create tuples and publish them.
```

```
13 my @field_names = ('Symbol', 'Price');
14 # Loop "forever"...
15 for(my $n = 0; ; $n++) {
16     # create tuple
17     my $tuple = C8::Tuple->new(@field_names);
18
19     # Set tuple fields, e.g. to "Symbol1", 1.
20     $tuple->fields('Symbol' . $n, $n);
21
22     # publish tuple to the stream
23     $publisher->write_tuple($tuple) or last;
24
25     sleep(1); # sleep a little bit
26 }
```

Lines 1-2 load the Sybase CEP modules C8::Tuple and C8::Publisher. The modules need to be available in the Perl include path. See Perl documentation for more details.

Lines 4-6 get the Sybase CEP Stream URI from the command line.

Lines 8-10 create Sybase CEP publisher and connect to the Sybase CEP Stream with the given URI.

Lines 12-13 declare the list of field names in the tuple.

Lines 15-26 comprise the main loop: create a tuple, publish it to Sybase CEP, sleep.

Lines 16-17 create a tuple with the given list of fields.

Lines 19-20 set tuple fields to the given values.

Lines 22-23 publish a newly created tuple.

Line 25 instructs the adapter to sleep.

Perl Output Adapter (Receiving Data from a Sybase CEP Stream)

An example that demonstrates how a simple output adapter subscribes to messages in the Sybase CEP stream.

The following example receives data from a Sybase CEP stream and publishes that data.

The full listing with line numbers is below:

```
1  use C8::Subscriber;
2
3  # Get command-line arguments.
4  my $usage = "$0 <uri>";  # Usage message to use if err
5  my $uri = shift or die "ERR: uri is not specified.\nUsage: $usage
\n";
6
7  # Create a subscriber object.
```

```
8  my $subscriber = C8::Subscriber->new();
9  $subscriber->connect($uri) or die "ERR: cannot subscribe to
'$uri'\n";
10
11 # First, print column names.
12 print join(',', $subscriber->column_names()) . "\n";
13
14 # then print each message row
15 while(1) {
16    my $tuple = $subscriber->read_tuple or last;
17    print "Ts:     " . localtime($tuple->timestamp) . "\n";
18    print "Fields: " . join(',', $tuple->fields) . "\n";
19 }
```

Line 1 loads the Sybase CEP module C8::Subscriber. The module must be available in the Perl include path. See Perl documentation for more details.

Lines 3-5 gets the Sybase CEP Stream URI from the command line.

Lines 7-9 creates Sybase CEP subscriber and connect to the Sybase CEP stream with the given URI.

Lines 11-12 get the list of column names from the Sybase CEP Server and print it out.

Lines 15-19 comprise the main loop.

Line 16 reads the new tuple.

Lines 17-18 prints the tuple.

Installation and Configuration

This section explains how to run the Sybase CEPPerl SDK installation script and how to specify the path to the library files.

Running the Perl Installation Script

Before you use the Sybase CEP Perl SDK, run the installation script, named `Makefile.PL`.

On Microsoft Windows, this file is typically in the directory:

```
C:\Program Files\SybaseC8\Server\sdk\perl\SybaseC8
```

On UNIX-like operating systems, this file is typically in the directory:

```
/home/<userID>/sybasec8/server/sdk/perl/SybaseC8
```

If you are installing modules into the site-specific perl library directories, you will need root privileges on UNIX-like operating systems or /administrator privileges on Microsoft Windows.

If you are installing into your own personal perl libraries, you do not need root privileges, but you need to install the libraries in such a way that perl can find them. To install into personal directories, you need a command similar to the following:

```
perl Makefile.pl LIB=/my/dir/perllib \
INSTALLMAN1DIR=/my/dir/man/man1 \
INSTALLMAN3DIR=/my/dir/man/man3 \
INSTALLBIN=/my/dir/bin \
INSTALLSCRIPT=/my/dir/scripts
```

Specifying the Path to the Library Files

Download and install the Perl modules that the Sybase CEP SDK relies on.

Use the commands shown below:

```
1   $ perl -MCPAN -e shell
2   > install URI::URL LWP::UserAgent Text::CSV  \
  SOAP::Lite XML::DOM XML::XPath Error
3   > quit
```

Due to formatting constraints, this is shown as 4 lines rather than 3; the 2nd and 3rd lines should be a single line without a backslash.

Line 1: Execute a command that goes to CPAN (Comprehensive Perl Archive Network).

Line 2: When prompted, enter the names of the libraries to be downloaded from CPAN.

Line 3: Exit the perl interpreter.

Note that these commands should be entered when you are prompted to enter commands. Depending on how your system is initially configured, you may be asked various questions before you are shown the prompts for these two commands.

To use the Sybase CEP Perl API, the directories that contain the Sybase CEP Perl modules must be in the Perl #include path. The Sybase CEP Perl modules are in the subdirectories:

```
server/sdk/perl/SybaseC8
server/sdk/perl/SybaseC8/SybaseC8
```

under the directory in which you installed the Sybase CEP product.

On Microsoft Windows, these are typically:

```
C:\Program Files\SybaseC8\Server\sdk\perl\SybaseC8
C:\Program Files\SybaseC8\Server\sdk\perl\SybaseC8\SybaseC8
```

On UNIX-like operating systems, these are typically:

```
/home/<userID>/sybasec8/server/sdk/perl/SybaseC8
/home/<userID>/sybasec8/server/sdk/perl/SybaseC8/SybaseC8
```

Prior to loading the module, you will need to do at least one of the following:

- Execute the command:

```
use lib "/my/dir/perllib"
```

- Set the PERL5LIB env variable.
- Use perl's -I switch.

Running the Perl example

This section describes the steps and commands you need to run the Perl example.

To run the example that Sybase CEP supplies, perform the following actions:

1. Start Sybase CEP Server.
2. Start Studio and create an input stream that has the following schema:

Field Name	Data Type
Symbol	String
Price	Float

3. Copy the URI of this stream. To copy the URI, go to the Sybase CEP Studio window, select the Explorer View, then click on the stream, and then look at the Properties tab for the stream and copy the URI shown in the Stream URI field.

   Alternatively, you can see the stream URI in the top of the stream viewer window. See *Acquiring the Address (URI) of a Stream* on page 150 for more information.
4. Execute the following command to start the publisher:

```
perl -I C8 examples/c8_publish.pl <URI just copied>
```

5. Execute the following command to start the subscriber:

```
perl -I C8 examples/c8_subscribe.pl <URI just copied>
```

   In this over-simplified example, the subscriber is actually reading from the input stream. We did this to minimize the number of steps that you would need to execute before you could see a publisher and subscriber working. In practice, you must also:
6. Create an output stream.
7. Look up the URI of that output stream.
8. Launch the `subscribe.pl` script with the URL of the output stream rather than the URL of the input stream.
9. Write one or more queries that fed data into the output stream.

Troubleshooting the Perl SDK

This section provides troubleshooting tips that are specific to the Perl SDK.

If you get an error similar to the following:

```
Can't locate SybaseC8/Publisher.pm in @INC (@INC contains: /usr/lib/
perl5/5.8/cygwin /usr/lib/perl5/5.8 /usr/lib/perl5/site_perl/5.8/
cygwin /usr/lib/perl5/site_perl/5.8 /usr/lib/perl5/site_perl/5.8/
cygwin /usr/lib/perl5/site_perl/5.8 /usr/lib/per l5/vendor_perl/5.8/
cygwin /usr/lib/perl5/vendor_perl/5.8 /usr/lib/perl5/vendor_perl/
5.8/cygwin /usr/lib/perl5/vendor_perl/5.8 .) at examples/
sybasec8_publish.pl line 10. . . .
```

The most likely cause is that you have not set your Perl include path to include the Sybase CEP directory that includes the Perl modules. To configure your environment properly, see *Prerequisites* on page 303.

If you get an error similar to the following:

```
Can't locate URI/URL.pm in . . .
```

The most likely cause is that you have not set your path to include the appropriate CPAN modules. For a list of these modules, see *Specifying the Path to the Library Files* on page 309. If you have not acquired these modules, you may need to do so. If you have acquired them, you need to set your perl include path to include these modules.

Additional troubleshooting tips are in *Troubleshooting* on page 351

Sybase CEP PythonPython SDK

This chapter provides a brief explanation of the Sybase CEP Python SDK.

The Sybase CEP Python SDK enables you to do the following:

- Write an input adapter or an output adapter in Python.
- Start or stop the Sybase CEP Server from inside a Python program.

Python API Interface

This section describes how adapters typically behave, and how they interact with the Python API Interface.

A typical adapter performs the following tasks:

1. Acquiring a URI) that will uniquely identify a specific stream and telling the communication layer provided by Sybase CEP how to find that stream.
2. Opening a connection to the stream.
3. Writing the desired data. Typically the write operation is in a loop; the adapter will continually send multiple rows of data.
4. Closing the connection.

When you use the Python SDK, the URI of the data stream is typically passed to the program as a command-line parameter. To find the URI of an active stream, look at the stream properties page in Sybase CEP Studio. Right-click on the stream, then choose "properties", and read the stream's URI in the Stream URI.

When you use the Python SDK, there is no explicit call to close the connection. The connection is closed when you destroy the connection object.

We recommend that you use Version 2.4.2 or later of Python.

Python API

This section describes the three parts of the Python API: the tuple, publisher, and subscriber classes.

The Python API consists of three parts:

- A Tuple class for manipulating Sybase CEP messages.
- A Publisher class for sending data to Sybase CEP Server.
- A Subscriber class for receiving data from Sybase CEP Server.

With this combination, you can write input or output adapters. For example, in an input adapter, you read data from the data source, transform it into an appropriate format, and then

use a Publisher object to send the data to the server. Similarly, in an output adapter, you use a Subscriber object to read data from the server, then transform the data into the external data format and write the data to its destination.

The complete Sybase CEP Python SDK, including example source code, is available in the sdk/python subfolder of your Sybase CEP Server installation. The primary files are:

- `sdk/python/c8.py`, which contains the Tuple, Publisher, and Subscriber classes.
- `sdk/python/examples/c8-pubsub.py`, which contains example code using a Publisher and a Subscriber.
- `sdk/python/examples/c8-server.py`, which contains example code to start and stop Sybase CEP Server.

Tuple

A Tuple object represents a tuple, or in other words, a row of data.

If you are writing data to a stream, you create a tuple object, fill it with data, then write it to the stream. Similarly, if you are reading data from a stream, you receive a row of data, read the individual values from the tuple object, and then write the data to your chosen destination.

- **getcolumnnames() –** Returns a list that contains the names of the tuple columns.
- **gettimestamp() –** Gets the tuple's timestamp, expressed as the number of seconds (not microseconds) since midnight January 1, 1970.

  The time module in the Python library contains functions to convert between seconds and more human-readable formats such as "YYYY-MM-DD HH24:MI:SS".
- **getvalue(self, name) –** Gets the value in the field specified by the name parameter.
- **setvalue(self, name, value) –** Sets the value in the field specified by the name parameter to the value specified in the value parameter.
- **getvalues(self) –** Gets the value in the field specified by the name parameter.
- **setvalues(self, value) –** Sets the value in the field specified by the name parameter to the value specified in the value parameter.
- **getcsv(self) –** Returns the tuple as a single CSV string.The first field in the CSV string is the row timestamp value in microseconds since 00:00:00 Jan 1, 1970.
- **setcsv(self, str) –** Given a string that contains comma-separated values, set the values of the fields in the tuple to the values in the string. Note that the first field in the CSV string should be the row timestamp value in microseconds since 00:00:00 Jan 1, 1970.

Publisher

Create a Publisher object to write data to a stream. You need one Publisher object per stream.

- **write_tuple(self, tuple) –** Writes a tuple object to the stream.

  There is no explicit call to close the connection. The connection will be closed when you destroy the C8::Publisher object.

Subscriber

Create a Subscriber object to read data from a stream. You need one Subscriber object per stream.

- **getcolumnnames() –** Gets the list of column names. Valid only after successful connect call.

  There is no explicit call to close the connection. The connection will be closed when you destroy the Subscriber object.
- **read_tuple() –** Gets the next message from the stream. The first row returned is always the title row, which contains the names of the fields. This is true even if you connect to a stream that transmits messages to other subscribers. (A stream may have more than one subscriber.)

  Note: Calls to read_tuple() are blocking calls. The call will not return until a message is received on the stream.

  Returns the tuple object or an undefined value if there are no more messages in the stream or a connection error occurs.

Python Input Adapter (Sending Data to a Sybase CEP Stream)

An example that demonstrates how a simple publisher writes messages to a Sybase CEP stream. This could be extended to make it an input adapter.

The stream schema is assumed to have exactly two fields ('Symbol' and 'Price') and the generated messages look like the following:

Row 1, 1

Row 2, 2

Row 3, 3

...

```
import sys
import time
from threading import Thread
from c8 import C8
# When you start this program, pass:
# 1) the URI of the stream to publish to, for example,           #
"ccl://localhost:6789/Stream/Default/Subscriber2/StreamIn1"
# 2-N) the column names of the stream
class C8Test(Thread):
def __init__ (self, mode, uri, column_names):
 Thread.__init__(self)
 self.mode = mode
```

```
 self.uri = uri
 self.column_names = column_names

def run(self):
 if self.mode == 'publish':
   self.publish()

def publish(self):
   publisher = C8.Publisher(self.uri)
   j = 0
   while True:
     tuple = C8.Tuple(self.column_names)
     for i in range(len(self.column_names)):
         tuple.setvalue(self.column_names[i], self.column_names[i]
         + '-' + str(j))
     # print 'Pub Ts:  ' + repr(tuple.gettimestamp())
     # print 'Pub Sym: ' + tuple.getvalues()[self.column_names[0]]
     print 'Pub Csv: ' + tuple.getcsv()

     publisher.write_tuple(tuple)
     time.sleep(1)
     j = j + 1
def main():
try:
 if len(sys.argv) < 3:
   print 'Error: Wrong number of arguments, usage: ' + sys.argv[0]
         + ' <uri> <column-name>, [<column-name>, ...]'
   return

 # Extract column names from the command-line arguments.
 columns = [ ] + sys.argv;
 # Ignore/remove the name of the program (sys.argv[0])
 columns.pop(0);
 # Ignore/remove the uri (sys.argv[1])
 columns.pop(0);
 pub = C8Test('publish', sys.argv[1], columns)
 pub.start()
 # This example runs for 60 seconds.  A real
 # adapter would probably run indefinitely.
 while True:
   time.sleep(60)

except IOError (errno, strerror):
 print "I/O error(%s): %s" % (errno, strerror)
else:
 print 'bye!'
if __name__ == '__main__':
main()
```

Python Output Adapter (Receiving Data from a Sybase CEP Stream)

The following example demonstrates how the Python output adapter reads data in a Sybase CEP stream.

```
from c8 import C8
# This is the URI to which we will try to connect.
uri = "ccl://localhost:6789/Stream/Default/Subscriber2/StreamOut1"
# Create a new Subscriber object.
subscriber = C8.Subscriber(uri)
# Minimize buffering (do this only for demo).
# This reduces performance, but makes sure that there
# is not much delay before the output is visible.
subscriber.buf_size = 1
# Retrieve several rows.
j = 1
while j <= 6:
tuple = subscriber.read_tuple()
print 'Sub Csv: ' + tuple.getcsv()
j = j + 1
```

Configuring your Python environment

Make sure that the python interpreter can find the `c8.py` file.

The `c8.py` file can be found in the following location:

```
server/sdk/python/c8.py
```

Adapter Definition Language

This chapter describes the Adapter Definition Language (ADL), which is an XML-based language that can describe the parameters of an in-process adapter. Out-of-process adapters do not use ADL.

You only need to read this chapter if you are writing an in-process adapter.

Adapter Definition Language: Overview

Use the ADL to modify how your adapters interact with Sybase CEP Studio.

An in-process adapter may have zero or more adapter properties. These allow one instance of an adapter to have custom values that tell it to behave differently (such as use a different data source) than other instances of the adapter. For example, the ReadFromCSVFile adapter, which is one of the built-in adapters supplied by Sybase CEP, allows you to specify which file to read from. Similarly, if you write your own in-process adapter, you may use parameters to specify the exact data source to read from. If you have not already looked at any built-in in-process adapters, we recommend that you look at one now. Create a stream, attach the ReadFromCSVFile adapter to that stream, and then click on the adapter in the Explorer View of Sybase CEP Studio. The Explorer View of Sybase CEP Studio will display the adapter properties, such as Filename, Loop count, Rate, and so on.

ADL allows you to describe your parameters in such a way that Sybase CEP Studio can display labels and empty fields, and then pass the filled-in values to your in-process adapter. For example, your adapter might use ADL to tell Sybase CEP Studio to display the label "Filename", read the string that a user types into a field next to that label, and then pass your string to you.

As we mentioned in the chapter on creating an in-process adapter, your adapter will consist of two files: one file is the library with your `Initialize()`, `Execute()`, and `Shutdown()` functions, and a second file (the .adl file) contains a description of the adapter's parameters. This chapter describes what that .adl file will contain.

Each ADL File must include:

- Name of the Adapter
- Direction of Adapter: Input or Output
- ScreenName
- Vendor
- Version
- Description
- InstanceParameter Definitions

Each InstanceParameter Definition comprises one or more parameter attributes that each contain:

- Name
- Type
- Default Value
- Description
- A flag to indicate whether the parameter is required or not.

The ADL file must also include Adapter Definition information such as the xmlns field. For details, see the "AdapterDefinition" section in the example below.

Below is an example of an ADL file :

```
<AdapterDefinition xmlns="http://www.sybase.com/adl/2005/04/"
Name="ReadFromXmlFileAdapterType"
xmlns:ns1="http://www.sybase.com/cpx/2004/03/"
Type="ns1:AdapterType" Direction="Input"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
SupportsGuaranteedDelivery="true">
<ScreenName>Read From XML File Adapter</ScreenName>
<Vendor>Sybase, Inc.</Vendor>
<Version>1.0</Version>
<Description>An input adapter that reads tuples from an XML file
</Description>
<LibraryName>sdk_adapter_lib</LibraryName>
<InitializeFunction>my_input_adapter_initialize</
InitializeFunction>
<ExecuteFunction>my_input_adapter_execute</ExecuteFunction>
<ShutdownFunction>my_input_adapter_shutdown</ShutdownFunction>
<ReconnectFunction>my_input_adapter_reconnect</ReconnectFunction>
<InstanceParameterDefinitions>
<Parameter Name="Filename" xsi:type="xsi:string">
<Default/>
<Description>The name of the file from which to read tuples.
</Description>
<Required>1</Required>
</Parameter>
<Parameter Name="Rate" xsi:type="xsi:integer">
<Default/>
<Description>Post tuples at given rate (tuples per second)
</Description>
<Required>0</Required>
</Parameter>
</InstanceParameterDefinitions>
</AdapterDefinition>
```

When Sybase CEP releases updated versions of the software, the structure of ADL files may change. If you want to create your own ADL file, and if you want to use an existing ADL file (such as `c8_read_from_csv_file.adl`) as a baseline, you should use the version that came with your product, which may differ from the version shown above. The ADL files supplied by Sybase CEP can be found in the directory

```
<install_dir>/Studio/plugins
```

On Microsoft Windows, this is typically:

```
C:\Program Files\sybasec8\Studio\plugins
```

On UNIX-like operating systems, this is typically:

```
/home/username/sybasec8/studio/plugins
```

The ADL file begins with a section similar to that shown below:

```
<AdapterDefinition
xmlns="http://www.sybase.com/adl/2005/07/"
Name="UserDefinedReadFromCsvFileAdapterType"
xmlns:ns1="http://www.sybase.com/cpx/2004/03/"
Type="ns1:AdapterType"
Direction="Input"
SupportsGuaranteedDelivery="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
<ScreenName>CServer SDK Adapter: Read From CSV File</ScreenName>
<Vendor>Sybase, Inc.</Vendor>
<Version>1.0</Version>
<Description>An input adapter that reads messages from a CSV file
</Description>
<Documentation>An input adapter that reads messages from a file that
            contains messages in the Comma Separated Values (CSV)
            format, one message per line
</Documentation>
```

Change the Name attribute to an appropriate value. Do not use spaces or tabs in the name.

Warning! Multiple ADL files with the same Name attribute in the AdapterDefinition will result in an error message.

The Direction="Input" attribute may require modification to Direction="Output" for an output adapter.

Specify SupportsGuaranteedDelivery="true" if yours is a Guaranteed Delivery adapter. See *Implementing Guaranteed Processing* on page 89 for more information.

The ScreenName is the name displayed by Studio in the drop-down selection menu that you see when you attach an adapter. Use a name that users will understand. While multiple in-process adapters are able to display the same screen name, users will be unable to differentiate between identically displayed adapter screen names.

The Description and Documentation attributes provide summary and detailed information relevant to the adapter.

To invoke your C/C++ code, the Sybase CEP Engine needs to know the names of the `Initialize()`, `Execute()`, `Shutdown()`, and possibly `Reconnect()` functions and the name of the library file that contains those functions. These are provided by:

```
<LibraryName>
sdk_adapter_lib
</LibraryName>
<InitializeFunction>
my_input_adapter_initialize
</InitializeFunction>
<ExecuteFunction>
my_input_adapter_execute
</ExecuteFunction>
<ShutdownFunction>
my_input_adapter_shutdown
</ShutdownFunction>
<ReconnectFunction>my_input_adapter_reconnect</ReconnectFunction>
```

The sdk_adapter_lib is the name of the loadable library containing the three entry points. The library name does not contain a trailing ".dll" or ".so". This permits cross-platform loading of libraries that will be handled by the Sybase CEP Engine.

The external name of each of the adapter entry points specifies the name that Sybase CEP Engine will call for initialization, execution, shutdown, and reconnection (for Guaranteed Delivery only) of the adapter. The name must match the name in the C/C++ code that you wrote. The name must also be externalized within the library.

The parameters section is the next section of the ADL file.

Each parameter will be displayed on the adapter's Properties View in Sybase CEP Studio. The exact form of the display depends on each parameter's definition. An example of a parameters section is shown below.

```
<InstanceParameterDefinitions>
<Parameter Name="Filename" xsi:type="xsi:string">
<Default>stock-trades.csv</Default>
<Description>The file from which to read rows.</Description>
<Required>false</Required>
<ScreenName>Filename</ScreenName>
<Documentation>
  The name of the file from which to read rows.
</Documentation>
<MultiLine>false</MultiLine>
</Parameter>
<Parameter Name="Port" xsi:type="xsi:integer">
<Default/>
<Description>TCP port number of the message source</Description>
<Required>true</Required>
<ScreenName>Port</ScreenName>
<Documentation>TCP port number of the message source</Documentation>
<MultiLine>false</MultiLine>
<Min>1</Min>
```

```
<Max>65535</Max>
</Parameter>
```

The Parameter attribute configures an individual parameter. There can be any number of parameters in an ADL file; the exact number depends on your needs. The Default, Description and Required attributes are mandatory for each parameter. The remaining attributes are optional.

The Name attribute is used by the C/C++ code to retrieve the displayed value of the parameter. Each parameter name must be unique within an ADL file.

The xsi:type attribute determines the data type of the parameter. See *Printing a Parameter* on page 187 for a table that shows which xsi:type corresponds to each internal Sybase CEP datatype.

The Default attribute determines the default value of the parameter. This value gets displayed in Sybase CEP Studio on the initial display. When the value is modified, Sybase CEP Studio remembers the modified value so it does not have to be reset on subsequent Sybase CEP Studio invocations. The Default must be of the same data type as the xsi:name, otherwise Sybase CEP Studio will issue an error message. Notice that in this example *Port* does not have a default and the Required attribute is true, meaning you must enter a value.

The Description provides textual information for the "?" popup help message Sybase CEP Studio displays.

The Required attribute specifies whether it is absolutely necessary for you to provide a value for this parameter. If you provide a Default for a Required parameter, and do not modify the parameter, then the default will be used. If you blanks the input field of a Required parameter, Sybase CEP Studio will highlight the input field to indicate that you must input a value. An example of a possibly required parameter is a filename for an output adapter that writes to a file.

The ScreenName value gets displayed to the left of the input field. The ScreenName prompts you for inputting into the field.

The MultiLine attribute causes a larger rectangular display to appear on the adapter's Properties View in Sybase CEP Studio. This would be used for parameters requiring potentially lengthy input strings. An example is a database query that consumes several lines of input.

Numeric datatypes may use the Min and/or Max attributes. It is possible to have a Min without a Max and vice versa; these attributes are independent.

Note: The numeric value of the Port parameter in the above example must be between 1 and 65536.

Warnings and Tips

This section contains tips for creating your own ADL files.

- When you create your own ADL file, make sure that you change only values, and not tags, inside the ADL file.
- Do not edit the AdapterDefinition section except for the direction and the name of the adapter.
- Certain characters, such as "<" and ">", have special meanings in XML and should not be used in the values inside an ADL file. If you need to use one of these characters as part of a value, the best solution is to surround it by special delimiters that tell the XML processor to treat the characters as literals rather than as part of the language. The delimiters are "<![CDATA[" and "]]>". An example is shown below:

  ```
  <Documentation>
      <![CDATA[
     Inside these delimiters, the following characters will be
     treated as literals rather than as XML language elements:
        < > & ' "
     (i.e. the greater than sign, less than sign, ampersand,
     single quote, and double quote).
      ]]>
      </Documentation>
  ```

  For a complete and up-to-date list of special characters, see the XML standards.

  ```
  XML 1.0: http://www.w3.org/TR/2004/REC-xml-20040204/
  XML 1.1: http://www.w3.org/TR/2004/REC-xml11-20040204/
  ```

  In particular, see sections 2.2 and 2.4 of each.
- When you upgrade yourSybase CEP Engine, the Sybase CEP Studio installation program deletes and then re-creates the directory that has the ADL files. Any ADL files you have added will be lost. Each ADL file that you create should be backed up in a location outside your c8 directory.
- When you upgrade, you should not copy your backed-up ADL files to your new directory; you should instead create new ADL files based on the structure of the ADL files in your upgraded Sybase CEP system, but with the same types of customizations as were in the backed-up ADL files that you built yourself.

Configuring Your System to Find ADL Files

This section describes the server process for searching for .adl files, and where to find the .adl file directories on different operating systems.

ADL files are always assumed to be in a plugins directory. This is the server's search path for .adl files:

- The plugins directory specified by the SybaseC8/General/PluginsFolder preference in the server's configuration file. Unless you change this preference or install in a directory other than the default directory, the plugins directory on Windows platforms is:

  ```
  C:\Program Files\SybaseC8\Server\plugins (on Microsoft Windows)
  ```

  ```
  On UNIX-like operating systems, the plugins directory is:
  ```

  ```
  $HOME/sybasec8/server/plugins
  ```
- <app path>/plugins
  (in other words, in the plugins subdirectory of the directory in which the application was started)
- <app path>/../plugins
  (in other words, in the plugins directory that is a sibling of the directory in which the application was started)
- ./plugins
  (in other words, in the plugins subdirectory of the current directory)
- ../plugins
  (in other words, in the plugins directory that is a sibling of the current directory)
- $C8_PLUGINS_FOLDER
  (or, in Microsoft Windows, the equivalent environment variable: %C8_PLUGINS_FOLDER%)
- The user plugins directory.
  On Microsoft Windows, this is:
  `SybaseC8Repository\<version>\plugins`
  On UNIX-like operating systems, this is:
  `SybaseC8Repository/<version>/plugins`
  (Note the blank space in the path name on both operating systems.)

Studio (and the compiler) normally look for .adl files in:

```
C:\Program Files\SybaseC8\Studio\plugins (on Microsoft Windows)
```

```
or
```

```
$HOME/sybasec8/server/plugins (on UNIX-like operating systems)
```

If the CCL compiler is unable to find the .adl files after searching the directories specified above, the compiler issues a message stating that it cannot locate the plugins directory.

Importing New Adapters

This section explains the process of adding a custom adapter to the Attach Adapters Dialog list of out-of-the-box Sybase CEP adapters.

In Sybase CEP Studio, the Attach Adapters Dialog displays a list of input and output adapters that come with Sybase CEP. When developing a new in-process adapter, you will typically want it to be included in the list of available adapters that you can attach. To do this, either copy the .adl file to Sybase CEP Studio's plugins directory and then re-start Sybase CEP Studio, or, if you don't want to stop and re-start Studio, import the ADL file into Sybase CEP Studio.

To import a new Adapter Definition, go to the Tools menu and select "Import Adapter Definition..." This brings up a list of ADL files that can be selected to import. Select the ADL file for your new adapter. Once successfully imported, the Adapter will be available for selection in the list of Adapters in the Attach Adapter Dialog.

Server Plugins

Extend Sybase CEP Server's functionality with one or more plugins.

Each plugin is a linkable library, either a .so file on UNIX-like operating systems, or .dll on Microsoft Windows.

Before you read the rest of this chapter, read *Plugins* on page 44 which provides an overview of how plugins work.

In the next few sections, we will describe:

- The types of events that the server plugins can handle.
- The generic plugin that Sybase CEP provides.
- How to use the generic plugin to implement High Availability (HA) Sybase CEP Server managers.
- Authentication-related plugins.

Message-Driven Plugins

A plugin is an event handler. When an event occurs, specific code within the handler is invoked.

That code can, in turn, perform the action that is required. For example, if the event is that the original manager died and the backup has taken over as manager, then the plugin might send an email notifying a system administrator that a manager died and a new backup manager must be started. A High Availability Server includes 1 or more containers, which do work such as executing queries, and 1 or more managers, which assign particular work to particular containers. As another example, the plugin may call a netsnmp process to send a "trap" signal to an event handler in another process outside the Sybase CEP manager.

Sybase CEP Server Events

Sybase CEP Server's manager process writes most messages reflecting the Server's status to special data streams called server status streams.

Every message appears as a row in a server status stream and contains the following information:

- SourceTimestamp: The message timestamp.
- MessageGroup: The group to which the message belongs.
- ObjectID: The object identifier for the group.

Events

The manager writes status information to Server Status Streams, which are special Sybase CEP streams that transmit messages with various server events and metrics.

Every message appears as a row in a Server Status Stream and contains the following information:

SourceTimestamp: The message timestamp.

MessageGroup: The group to which the message belongs.

ObjectID: The object identifier for the group.

The code to listen to the status stream and act upon the events is called a "plugin", and it runs as part of the server process.

There are two groups of status events: container events that apply to the container server and manager events that apply to the manager server. Examples of status events include CclApplicationInfo Status messages, CclQueryInfo Status messages, ContainerInfo Status messages, and ManagerInfo Status messages.

Container Events

A description of status events for containers, including their message codes and frequencies.

The status events currently implemented for containers are:

Message Code	Frequency	Description
ServiceStarted	On event	Event: service is started (value = 1)
ServiceStopped	On event	Event: service is stopped (value = 1)
ServiceKilled	On event	Event: service is killed via soap (value = 1)
ContainerTotalMemory	1/sec	Total memory available to Sybase CEP Server in bytes)
ContainerUsedMemory	1/sec	Total memory used by the Sybase CEP Server in bytes
ContainerCPUUtilization	1/sec	Percentage of CPU utilized by the Sybase CEP Server process (1 = 100%)
ContainerCPUTime	1/sec	CPU Time used by the Sybase CEP Server process since start in microseconds

Message Code	Frequency	Description
ModuleSentMessages	1/sec	Count of sent messages (value = count)
ModuleReceivedMessages	1/sec	Count of received messages (value = count)
ModulePendingMessages	1/sec	Count of pending messages (value = count)
ModulePersistentDbPendingMessagesNum	1/sec	Count of persistent db pending messages (value = count)
ModuleError	On event	Event: module generated an error message (value = error ModuleRunState xml or text)
ModuleRunState	On event	Event: module is started (value = 1) or stopped (value = 0)

Pending messages are messages that have been received but not yet processed.

Manager Events

A description of status events for managers, including their message codes and frequencies.

Status events currently implemented in managers are:

Message Code	Frequency	Description
ServiceStarted	On event	Event: service is started (value = 1)
ServiceStopped	On event	Event: service is stopped (value = 1)
ServiceKilled	On event	Event: service is killed (via soap) (value = 1)
ManagerHAPromotedToPrimary	On event	Event: manager HA node promoted to primary (value = 1)
ManagerHADemotedToBackup	On event	Event: manager HA node demoted to backup (value = reason)
ManagerHAParticipatingInElection	On event	Event: manager HA participating in primary elections (value = 1)

ContainerAdded	On event	Event: Container added (value = "Active" or "Passive")
ContainerKilled	On event	Event: Killing container (value = reason)
ContainerRemoved	On Event	Event: Container removed (value = reason)
ProgramRegistrationState	On event	Event: program is registered (value = 1) or unregistered (value = 0)
ProgramLoadState	On event	Event: program is loaded (value = 1) or unloaded (value = 0)

CclApplicationInfo Status Messages

A description of runtime properties for projects.

The ObjectId column for this group is the full path to the project, in the following format: *workspace-name/project-name*, where *workspace-name* is the name of the workspace, and *project-name* is the name of the project.

CCLApplicationInfo messages include:

Event Name	Frequency	Description
AutoRestartAttempt	Whenever the project or the Server Container on which it is running auto-restarts.	Shows the number of times the project has auto-restarted.
CPUUtilization	Every second.	Shows the fraction of CPU the project utilizes, where a value of one (1) indicates 100% of total available processor CPU.
DroppedMessages	Every second.	Shows the total number of dropped out-of-order and late rows for the project.
InputMessages	Every second.	Shows the number of input rows the project has received. This is the sum of output rows for all external input streams.
LastError	Whenever the project generates an error.	Shows the most recent error the project has generated.

Event Name	Frequency	Description
OutputMessages	Once per second.	Shows the number of output rows the project has sent. This is the sum of input rows for all external output streams.
PendingMessages	Once per second.	Shows the number of pending rows the project's queries need to process. This is a sum of pending rows for all CCX modules in the corresponding program.
PendingPersistentMessages	Once per second.	Shows the number of rows the project needs to save to the disk. This is the sum of pending rows for all CCX modules in the corresponding program.
RunningTime	Every second.	Shows how long the project has been running.
RpcFailedMessages	Every second.	Shows the total number of failed RPC calls for the project.
RpcSentMessages	Every second.	Shows the total number of successful RPC calls for the project.
State	Whenever the state changes.	Shows a state change. State changes include: • Unregistered • Unregistering • Registered • Registering • RegisterFailed. • Started • Starting • StartFailed

CclQueryInfo Status Messages

A description of runtime message properties for specific CCL queries.

The ObjectId column for this group is the full path to the query, in the following format: *workspace-name/full-ccl-path/statement-number*, where *workspace-name* is the name of the workspace, and *full-ccl-path* and *statement-number* are a path and statement number assigned to the query by the compiler.

CCLQueryInfo messages include:

Event Name	*Frequency*	*Description*
InputMessages	Every second.	Shows the number of input rows the query has received.
OutputMessages	Every second.	Shows the number of output rows the query has sent.
RpcFailedMessages	Every five seconds.	Shows the total number of failed RPC calls for the query.
RpcSentMessages	Every five seconds.	Shows the total number of successful RPC calls for the query.

ContainerInfo Status Messages

This section describes runtime message properties for specific container processes.

The ObjectId column for this group is the container's URI, in the following format: *http://hostname:port* or *https://hostname:port*, where *hostname* and *port* are the host name and port number on which the container is running.

ContainerInfo messages include:

Event Name	Frequency	Description
ContainerAdded	Whenever a Container is added	Generates the following message: `Event: Container added (value= value)` where `value` is active when an active Container has been added, or passive when a passive Container has been added.

Event Name	Frequency	Description
ContainerKilled	Whenever a Container is killed	Generates the following message: `Event: Killing container (value=reason)` where *reason* is the reason the Container was killed.
ContainerRemoved	Whenever a Container is removed	Generates the following message: `Event: Container removed (value=reason)` where `reason` is the reason the Container has been removed.
CPUTime	Every second	Shows how much CPU time in microseconds the Sybase CEP Serverprocess has used since it has started.
CPUUtilization	Every second	Shows the fraction of CPU the Sybase CEP Server process is using, where a value of one (1) indicates 100% of total available processor CPU.
LogErrorCount	Every second	Shows the total number of errors that have been logged in the Server log for a given Container. This number is reset upon Container restart.
LogWarningCount	Every second	Shows the total number of warnings that have been logged in the Server log for a given Container. This number is reset upon Container restart.
TotalMemory	Every second	Shows the total bytes of memory available to Sybase CEP Server.
UsedMemory	Every second	Shows the total bytes of memory used by Sybase CEP Server.

ManagerInfo Status Messages

This section describes runtime message properties for specific manager processes.

Messages are generated on the primary active manager and any passive managers.

The ObjectId column for this group is the manager's URI, in the following format: *http://hostname:port* or *https://hostname:port*, where *hostname* and *port* are the host name and port number on which the manager is running.

managerInfo messages include:

Event Name	Frequency	Description
managerHA PromotedToPrimary	Whenever a new manager is promoted to primary status.	Generates the following message: `Event: manager HA node promoted to primary (value= reason)` where `reason` is the reason for the promotion.
managerHA DemotedToBackup	Whenever a manager is demoted to backup status.	Generates the following message: `Event: manager HA node demoted to backup (value= reason)` where `reason` is the reason for the demotion.
managerHA ParticipatingInElection	Whenever a manager participates in the election of a new primary manager.	Generates the following message: `Event: manager HA participating in primary elections (value= reason` where `reason` is the reason for the participation.

The Sybase CEP Generic Plugin

Sybase CEP provides a generic plugin that enables you to invoke a specified executable program when a specified status event occurs.

To use this plugin, update Sybase CEP Server's configuration file (`c8-server.conf` by default) to specify the event that you want to watch for and the name of the program that you want to execute when that event occurs. The relevant section of the configuration file looks like the following:

```
<section name="ManagerFailoverDDNSPlugin">
<preference name="LibraryName"
           value="c8_server_plugins_lib"/>
<preference name="InitializeFunction"
           value="c8_command_line_plugin_initialize"/>
<preference name="ExecuteFunction"
           value="c8_command_line_plugin_execute"/>
<preference name="ShutdownFunction"
           value="c8_command_line_plugin_shutdown"/>
<preference name="MessageGroup"
           value="ManagerInfo"/>
<preference name="MessageName"
           value="ManagerHAPromotedToPrimary"/>
<preference name="CommandName"
```

```
           value="REPLACE_WITH_PATH_TO_NSUPDATE"/>
<preference name="CommandArgument1"
           value="-k"/>
<preference name="CommandArgument2"
           value="REPLACE_WITH_PATH_TO_KEY_FILE"/>
<preference name="CommandArgument3"
           value="-v"/>
<preference name="CommandArgument4"
           value="REPLACE_WITH_PATH_TO_UPDATE_COMMANDS"/>
<preference name="MaxRunningProcesses" value="1"/>
<preference name="CommandTimeoutSeconds" value="30"/>
</section>
```

The section name ("ManagerFailoverDNSPlugin" in this case) may be any valid section name, but must be unique within this configuration file.

The first four preferences are required. You must specify:

- The LibraryName, which is the name of the library that contains the code. The c8_server_plugins_lib library is supplied by Sybase CEP).
- The names of the `initialize()`, `execute()` and `shutdown()` functions.

When you use the generic plugin supplied by Sybase CEP, the name of the library, and the names of the the `initialize()`, `execute()` and `shutdown()` functions are always the same.

The next preference, the EventType, is also required. Specify a value that is listed as an event type in the event tables shown earlier in this chapter.

The next preference, the CommandName, is also required. This is the name of the executable program that you want to run. The actual value depends upon what action you want to take when the event occurs.

Note: On UNIX-like operating systems, the full path must be specified as well as the program name. On Microsoft Windows, only the name of the program needs to be specified if the path to the program appears in the %PATH% environment variable.

The command argument parameters are command-line arguments to the program specified in the CommandName parameter. These are optional and will vary depending on what action you want to take and what parameters your command requires. You can have up to 1024 of these parameters.

The last two parameters, MaxRunningProcesses and CommandTimeoutSeconds, are also optional.

The MaxRunningProcesses parameter allows you to set an upper limit on the number of event-handling programs that are spawned by this instance of the plugin and are still executing. If you have already reached this limit and a new event arrives, the server checks which of the event-handling processes have been running for at least the amount of time specified by the CommandTimeoutSeconds parameter; if there are any, the server will try to kill one of those processes. If the server is unable to kill an old process, then it will not launch a new one. The default value for MaxRunningProcesses is 32.

The CommandTimeoutSeconds parameter should be used in conjunction with MaxRunningProcesses preference. If MaxRunningProcesses is reached, processes that have been running longer than the CommandTimeoutSeconds interval will be terminated, so that other processes can be spawned in their place.

Note: CommandTimeoutSeconds has no effect until MaxRunningProcesses is reached. The default value of CommandTimeoutSeconds is 60. A value of 0 means that the processes should not be timed out.

For events that occur once per second, most event-handling programs finish quickly. For example, if you want to update a display showing CPU consumption, the Nth update should finish before the N+1th update starts, which means that each update should take less than 1 second .

For events that occur infrequently the running time of event handling programs is less of a concern. However, you must have a good idea of the "worst-case scenario" so that you can set MaxRunningProcesses high enough. You may specify as many event/program pairs as you wish for each server. A single event can spawn multiple programs. You must specify a separate event/program pair for each of these as there is no direct way to specify that a single event will run multiple programs. Conversely, more than one type of event can spawn the same program. You'll need to specify a separate event/program pair for each of these as there is no direct way to specify that multiple events can spawn the same program. In each of these cases, you'll specify the parameters independently.

Server plugins may be used with either managers or containers. Make sure that you put the plugins section in the correct part of the server configuration file. Manager plugin information should go in the manager section and container plugin information should go in the container section.

Implementing Manager HA with the Sybase CEP Generic Plugin

One of the major uses of a plugin is to notify the DNS server of the IP address of the new primary manager node.

The new active manager must re-point the generic manager hostname to a new IP address by updating the record on the DNS server.

When the primary manager fails and the backup becomes the new primary manager, a ManagerHAPromotedToPrimary event will be triggered. The Sybase CEP generic plugin can be configured to respond to this event by calling a program (for example, nsupdate) that will update the DNS recordm such as nsupdates.

For more information on this topic, please consult the Sybase CEP *Installation Guide* .

Non-Message-Driven Plugins

The server may call a plugin's execute() function for reasons other than the arrival of a particular type of message in a status stream. In this section we describe plugins that are called without a status message being generated.

The User Authentication Plugins

Sybase CEP Engine Enterprise Edition enables users to explicitly permit or deny access to specified resources.

For example, you might allow only certain specific users or computers to subscribe to a stream that contains private financial data.

The Sybase CEP *Installation Guide* describes the Actions, Resources, and Subjects that may be restricted. In short, a subject is a user, a group, or a computer. A resource is a Sybase CEP Server object, such as a stream or a workspace. An action is a plugin action such as read, write, or create workspace. So, for example, there can be a rule created that allows user Jane Smith to subscribe and publish to the stream StockQuotes, while John Doe is allowed to subscribe to the stream but not to publish to it, and any other user is disallowed from reading or writing to the stream.

To implement this restriction, you can use a plugin that asks a user to enter her username and password before they is allowed to access a particular resource, such as subscribing to a stream. The plugin is passed the username and the password and then returns a value indicating whether the password matched the password for the specified user. If the password does not match, the server will not allow the user to access that particular resource. you can choose whatever software you wish to use to verify that the password is correct for the username.

Sybase CEP provides the following plugins for user authentication:

- LDAP: This plugin allows you to use standard LDAP as a way of determining whether a user is who they claim to be.
- htpasswd: This is a simple plugin that allows you to use encrypted password files to determine whether a user is who they claims to be.
- Pluggable Authentication Module (PAM): This is a tool that lets you use different authentication methods at different times, without re-compiling the program or recompiling the Sybase CEP Server.

In addition to using these plugins, you can write your own plugin, which can connect to some other system to verify the password. If you write your own plugin, specify the library name that contains the plugin, and the names of the `initialize()`, `execute()`, and `shutdown()` functions of the plugin. See the documentation of the htpasswd plugin for an example.

For any of these authentication plugins, you must customize the `c8-server.conf` file in at least two places. The first place is shown below; you must de-comment the "ACLFile" preference and, optionally, replace the file name "`c8-acl.xml`" with another file name if you want to use a different file:

```
<section name="SybaseC8/Security">
<section name="AccessControl">
<!-- The location of the Sybase CEP access control list file -->
<!-- Default: empty -->
<!-- <preference name="ACLFile"
   value="C:\Program Files\SybaseC8\Server/conf/c8-acl.xml"/> -->
```

You must, of course, customize the contents of the c8-acl.xml file. See the Sybase CEP *Installation Guide* for details.

The second part of the `c8-server.conf` file that you must customize will depend upon which plugin you are using.

For more information on this topic, please see the following:

- *User Authentication htpasswd Plugin* on page 338.
- *User Authentication LDAP plugin* on page 340.
- *User Authentication via Pluggable Authentication Module (PAM)* on page 341.
- The *Sybase CEP Installation Guide* , which explains how to configure the Sybase CEP Server configuration file (`c8-server.conf`) to tell the server to read the ACL file (`c8-acl.xml`).
- The Sybase CEP *Installation Guide* , which explains some contents of ACL files (Subjects, Resources, and Actions).

User Authentication htpasswd plugin

The htpasswd plugin uses an encrypted password file to authenticate a user and to determine whether the user is a member of any group that is permitted to access a particular resource.

As with any plugin, you must update the `c8-server.conf` configuration file to specify information about the plugin. Below is an excerpt from the default `c8-server.conf` file, which shows the preferences used by the htpasswd plugin, which has the 4 standard preferences (LibraryName, InitializeFunction, AuthenticateFunction, and ShutdownFunction) and 2 custom preferences (PasswordFilePath and GroupFilePath).

```
<section name="Plugin">
  <preference name="LibraryName"
     value="c8_server_plugins_lib"/>
  <preference name="InitializeFunction"
     value="c8_auth_plugin_htpasswd_initialize"/>
  <preference name="AuthenticateFunction"
     value="c8_auth_plugin_htpasswd_authenticate"/>
  <preference name="ShutdownFunction"
     value="c8_auth_plugin_htpasswd_shutdown"/>
  <preference name="PasswordFilePath"
     value="C:\Program Files\SybaseC8\Server/conf/htpasswd.txt"/>
```

```
  <preference name="GroupFilePath"
     value="C:\Program Files\SybaseC8\Server/conf/htgroup.txt"/>
</section>
```

The initialize and shutdown functions perform just as described in *Plugins* on page 44.

The AuthenticateFunction corresponds to the `execute()` function described in *Plugins* on page 44; this function is called each time a user gives her username and password in order to access a resource, such as a stream.

This custom entry PasswordFilePath contains pairs of usernames and passwords that the plugin uses to determine whether a user is who he or she claims to be.

Finally, the custom entry GroupFilePath lists the groups that each user is in. This is used when the ACL file permits or denies privileges to particular groups.

For example, suppose that you have multiple people who are system administrators, and we want to create a group named "SysAdmins" and allow any member of that group to create or destroy workspaces. To do this, perform the following actions:

1. In the ACL file (named `c8-acl.xml` by default), enter information similar to the following:

   ```
   <!-- Permit members of the group "sysadmin" to
        create and destroy workspaces -->
       <Rule RuleId="SysadminWorkspaceRul1" Effect="Permit">
     <Target>
      <Subjects>
        <!-- Any member of the "SysAdmin" group. -->
        <Group>SysAdmins</Group>
      </Subjects>
      <Resources>
        <!-- any workspace name.  (".*" is a regular
             expression that indicates any sequence of
              characters -- in other words, any name.)
        -->
        <Workspace>.*</Workspace>
      </Resources>
      <Actions>
        <CreateDestroy/>
        <GetStatus/>
      </Actions>
     </Target>
       </Rule>
   ```

   Group names, host names, and user names are not case-sensitive. Values may be regular expressions. In the example above, the "." represents any character, and the "*" is a repeat indicator which indicates that there may be any number of these characters.

2. In the groups file (we specified `htgroup.txt` in the GroupFilePath preference in the `c8-server.conf` file), enter at least the following:

   ```
   SysAdmins:jsmith
   ```

You can specify more than one member of a group, using the comma as a separator, for example:

```
SysAdmins:jsmith,pjones,andrews
```

3. Add an entry to the `htpasswd.txt` file which is specified in the PasswordFilePath preference of the `c8-server.conf` file. The password in this file must be encrypted using md5 format. Use apache's 'htpasswd -m' command to create htpasswd records. For example:

```
# Create a new htpasswd.txt file with a user named "root"
    # whose password is "carrot".
    htpasswd -bcm ./htpasswd.txt root carrot
    # Add user jsmith with password "taro9" to the htpasswd.txt
file.
    htpasswd -bm ./htpasswd.txt jsmith taro9
```

Note that the `htgroup.txt`file is a "plain text" file; no part of it is encrypted.

When a user attempts to create a new workspace, the following will happen:

- The user starts Studio and attempts to add a new workspace.
- The user enters the appropriate username and password when prompted.
- The server will then see the server configuration file that specifies the htpasswd plugin, and calls the function specified in the AuthenticateFunction. The server then passes the ID and password the user has entered.
- The plugin first encrypts the password and then looks in the htpasswd.txt file to determine whether the encrypted password matches the password in the file. The plugin then returns a value indicating thatthat the user has been authenticated.
- The server then retrieves the ACL file and sees that, although there is no rule explicitly permitting the user to create a workspace, the group "SysAdmins" is permitted to create a workspace.
- The server then looks in the htgroup.txt file and sees that the user is a member of the group "SysAdmins".
- The server then allows the authenticated user to create a new workspace.

User Authentication LDAP plugin

Update parameters in the `c8-server.conf` file to specify information about the LDAP plugin.

The LDAP plugin supplied by Sybase CEP follows the standard pattern for LDAP. The parameters that you must specify in the `c8-server.conf` configuration file are:

- Hostname: Required. For example, 'ldap.eng.c8.com.'
- Port: Optional. Default 389.
- Timeout: Optional. Connections timeout in sec. (default: 10).
- Version: Optional. 2 or 3 (default 3).

- Debug: Optional. Yes or no (default: no).
- SSL/TLS settings:
  - SslStart: Optional. Yes or no (default: no).
  - TlsStart: Optional. Yes or no (default: no).
  - TlsCheckServerCert: Optional. Yes or no (default: no).
  - TlsCaCert: Required if TlsCheckServerCert is yes. Path to the cert.

Note: The SSL/TLS settings are not available on Microsoft Windows.

- BindDN, BindPassword: Optional. If left empty/not specified then use anonymous search.
- User settings:
  - UserUidAttr: Optional. String (default 'uid').
  - UserBaseDN: Optional. String (for example, 'ou=People,dc=c8,dc=com').
  - UserFilter: Optional. String (for example, 'objectclass=posixAccount').
- Group settings:
  - GroupMemberUidAttr: Optional. String (default 'memberuid') - if not an empty string, then we provide user's uid (username) for groups search (also see GroupMemberDnAttr). This is typical setup for OpenLDAP and RedHat/Fedora Directory Server.
  - GroupMemberDnAttr: Optional. String (default '') - if not an empty string, then we provide user's DN for groups search (also see GroupMemberUidAttr). This is typical setup for Microsoft AD/ADAM Server.
  - GroupBaseDN: Optional. String (for example, 'ou=Groups,dc=sybase,dc=com').
  - GroupFilter: Optional. String (for example, 'objectclass=posixGroup').
  - GroupLimit: Optional. Integer - how many groups to return (default : 0 - no limit).

User Authentication via Pluggable Authentication Module (PAM)

Users of Linux and some other unix-like operating systems, including recent versions of Solaris, can use the Pluggable Authentication Module (PAM) system.

PAM allows PAM-compatible applications including the to switch authentication methods without recompiling the application. This allows a system administrator to upgrade to a completely different authentication system without recompiling the application. Changes are made by updating configuration files to specify which authentication method should be used and optionally by adding additional subroutine libraries to add new types of authentication methods. The additional subroutine libraries can be purchased, or you can develope one yourself.

Note: At the time PAM was added to Sybase CEP, PAM was not available on MS-Windows systems.

This document assumes that you already have the PAM modules that you need and therefore we explain only how to configure your to use existing PAM modules. For more information about PAM, including information about how to develop PAM modules of your own, see the PAM documentation on the internet. As of the date that this feature was introduced in Sybase CEP, these documents were located at:

```
http://www.kernel.org/pub/linux/libs/pam/
    Linux-PAM-html/Linux-PAM_SAG.html
http://www.kernel.org/pub/linux/libs/pam/
    Linux-PAM-html/Linux-PAM_ADG.html
```

The first of these documents is the System Installation Guide, which explains how to configure a PAM system when you already have the files you need. The second document explains how to write your own pluggable authentication module.

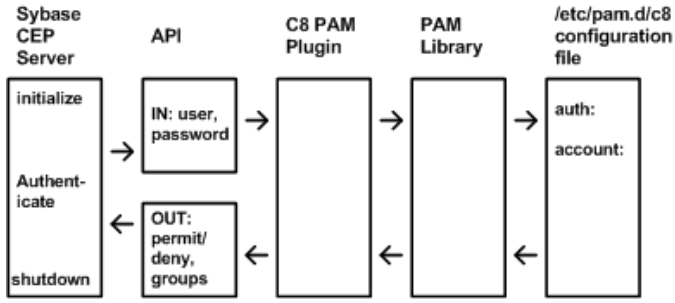
We assume:

- You have read an overview of the PAM system (for example, chapters 1-4 of the PAM System Administrator's Guide) and understand how PAM works.
- You have downloaded and installed the PAM files (library files, and so on) that you will need. Some unix-like operating systems may already come with PAM installed.
- You have acquired system administrator privileges on your computer or have other privileges sufficient to allow you to create or modify files under:
  `/etc`
  so that you can create or add to one of the following:
  - `/etc/pam.d`: A directory containing PAM configuration files.
  - `/etc/pam.conf`: A file containing PAM configuration information.

  Note: Access to the `/etc/pam.d` directory or the `/etc/pam.conf` file should be tightly restricted, otherwise, a malicious user could change the configuration files and weaken security.
- You have acquired system administrator privileges on your computer, or other privileges that allow you to create or modify the `c8-server.conf` file and the `c8-acl.xml` file.
  The location of the server configuration file depends on where you install to, but a typical location is:
  `<install-dir>/server/conf/c8-server.conf`

A Brief Overview of PAM with C8

In the diagram below, you can see the basic components involved when Sybase CEP uses PAM:



contains functions designed to work with PAM, including the initialize, authenticate, and shutdown functions. You will need to configure your `c8-server.conf` file to specify the name of the library that contains these functions.

The API box indicates what type of information is transmitted from the Sybase CEP Server to the PAM Plugin (which is a library that actually runs as part of the same process as the Server). The Server passes user and password information to the C8 PAM Plugin, and the C8 PAM Plugin returns a set of values that indicate:

- Whether the specified user is permitted or denied access.
- If the user is permitted access, then which groups the user is a member of.

The C8 PAM Plugin calls appropriate functions in the PAM library, and passes appropriate information to that library.

The PAM library does a variety of things, including:

- Reading the `/etc/pam.d/c8` file to determine which configuration method (for example password, retina scan, or fingerprint) should be used.
- Calling the appropriate functions to authenticate the user (for example, request password or fingerprint info).
- Returning a permit/deny value to the C8 PAM Plugin, which, in turn, passes this information back to the Sybase CEP Server.

The PAM library does more than this, but these are the key high-level tasks as far as the is concerned.

The `/etc/pam.d/c8` file specifies which authentication method should be used, and other details, which are explained in more detail below. For additional information, see the internet pages that document PAM.

Configuring Sybase CEP Server to Use PAM

As with any plugin, you must update the `c8-server.conf` configuration file to specify information about the plugin. The default `c8-server.conf` file included with your contains a commented-out section with PAM-related configuration parameters. The basic parameters are described here:

```
<!-- Sample PAM authentication plugin configuration -->
<section name="Plugin">
<preference name="LibraryName" value="c8_server_plugins_lib"/>
<preference name="InitializeFunction"
value="c8_auth_plugin_pam_initialize"/>
<preference name="AuthenticateFunction"
value="c8_auth_plugin_pam_authenticate"/>
<preference name="ShutdownFunction"
value="c8_auth_plugin_pam_shutdown"/>
<!--
 Uncomment the following line and change 'value' to
 set the password prompt that PAM sends to applications.
 The defaultprompt is "Password: ".  (Note the required
```

```
 space at the end.)  You should only need to do this if
 you encounter a system that does not use the default,
 and as such, it failing authentication.
-->
<!-- <preference name="PasswordPrompt" value="Password: "/> -->
</section>
```

Look carefully at your `c8-server.conf` file to see the exact parameters for your version of Sybase CEP. See also the .

By default, this section is commented out in the `c8-server.conf` file. Make sure that you de-comment it.

To use a PAM, you must specify:

- The name of the library file that contains the `initialize()`, `authenticate()`, and `shutdown()` functions.
- The name of the `initialize()` function in that library.
- The name of the `authenticate()` function in that library.
- The name of the `shutdown()` function in that library.

Since the C8 PAM library is supplied by Sybase CEP, the names of the library and the functions in it are known, so you can de-comment this section of the `c8-server.conf` file. You do not need to change the values of the library or function names.

Configuring the PAM Service

Create or edit a PAM configuration file to specify which authentication method (for example, password, or fingerprint) you want to use with . Sybase CEP Server calls the PAM library functions to read this configuration file. The PAM library functions then know which of the many PAM authentication functions to use. Complete documentation about this PAM configuration file can be found as part of the PAM documentation. Key aspects are covered below.

In this document, we assume that you will create

```
/etc/pam.d/c8
```

Although PAM allows 4 different types of management group rules (account, auth, password, and session), uses only two: account and auth.

- auth: This indicates that the user must be authenticated to prove that they are whom they claim to be, and allows the module to indicate which groups the user is a member of.
- account: This allows the system to permit or deny access based on factors other than the user's identity. For example, the module might temporarily deny a user access to a system if the system is already heavily loaded.

The examples below use only a single rule. See the PAM documentation if you need information about how to use a "stack" containing multiple rules.

This sample configuration file will check the user against the system password:

```
# type     control    module-path    module-arguments
# -------  --------   ------------   ------------------
auth       required   pam_stack.so   service=system-auth
account    required   pam_stack.so   service=system-auth
```

The type is either auth or account.

In this case, the user must follow the specified authentication rule, so we put required. If you are using a stack that contains multiple rules, you might specify values other than required for some rules. For more information, see the PAM documentation.

The module-path indicates which PAM library we want to use to perform the authentication. The module-path is either the absolute path and filename of the PAM to be used by the application (if the path begins with a '/'), or a relative pathname from the default module location: /lib/security/ or /lib64/security/, depending on the architecture.

The module-arguments are a space-separated list of tokens that can be used to modify the specific behavior of the given PAM. The exact values depend on which PAM library module you are using and can be found in the documentation for that specific module. In this example, we are specifying that we want the normal system-wide authentication method on this computer (typically a user ID and password).

This configuration file allows access to all users. This may be useful for testing but is not secure.

```
# type    control   module-path    module-arguments
# -----   --------  -------------  ----------------
auth      required  pam_permit.so
account   required  pam_permit.so
```

This configuration file denies all access and is useful for testing.

```
# type    control   module-path  module-arguments
# -----   -------   -----------  ----------------
auth      required  pam_deny.so
account   required  pam_deny.so
```

PAM Troubleshooting

This section provides troubleshooting tips for some PAM-related problems.

- Specifying the service:
  PAM allows two different file formats and locations for configuration files. If you use `/etc/pam.conf`, then you will must include a service column in the PAM configuration file. If you use `/etc/pam.d/c8`, then you must omit the service column from the PAM configuration file. For more information, see the documentation for PAM.
- PasswordPrompt:
  If your system uses a password prompt different from the one shown in the `c8-server.conf` file's PasswordPrompt preference which may occur if your password

prompt has been localized, then you need to change the `c8-server.conf` file's preference to match your actual prompt.

This is because at a certain point in the authentication process, the PAM library calls the Sybase CEP PAM plugin and passes the system's password prompt. The PAM library expects the to display the prompt, retrieve the password from the user, and then return the password to the PAM library. However, when Sybase CEP's PAM plugin receives the prompt, 's PAM plugin does not actually display the prompt, but instead returns password information that Sybase CEP Server already has. If the system's password prompt does not match the PasswordPrompt preference, then doesn't recognize when to return the password. To prevent this problem, set the PasswordPrompt preference to match your system's actual password prompt. Note that this string must exactly match the actual password prompt, including any blank space in the prompt.

- If the password prompt sent by the PAM library does not match the password prompt expected by the Sybase CEP PAM plugin, you do not get a detailed error message. The only symptom you can see is that authentication fails.

Datatype Mappings

This section describes the approximate mappings between CCL data types, ANSI data types, and Oracle data types.

Recommended datatype mappings between CCL and SQL

Not all data types can be mapped exactly. For example, CCL Integer is a 32-bit integer data type with a maximum positive value of 2,147,483,647. The closest data type in Oracle is either NUMBER(9), which can't hold some values that CCL INTEGER data type can hold, or NUMBER(10), which can hold some values that CCL INTEGER data type can't hold.

The following tables show the recommended mappings between CCL and SQL. These are useful if you are going to read data from a database server into your Sybase CEP Server or if you are going to write data to a database server from your Sybase CEP Server. For example, if you have an Oracle database server and you want to exchange BLOB data between your Oracle and Sybase CEP Servers, you should use VARCHAR(N) data type on the Oracle server with BLOB data type on Sybase CEP Server; however, if you are using a Sybase server then you should use TEXT on the Sybase server to correspond to BLOB on Sybase CEP Server.

Note: Some database servers (including) use LONG for a numeric data type, while other database servers (including Oracle) use LONG to refer to a character data type. Furthermore, some database servers (including Sybase CEP) use LONG for a numeric data type, while other database servers (including Oracle) use LONG to refer to a character data type.

These recommended mappings are based on our in-house testing. In some cases, other mappings will also work. For example, although we map Sybase CEP BOOLEAN to MySQL INTEGER, it is likely that MySQL BOOLEAN will also work.

Note: different database servers may have different length limits. Not all database servers allow VARCHAR(2147483647), so even when you are using the recommended type mappings you may still see truncation for long pieces of data.

Three tables are provided: The first two tables show the mappings between several specific database servers and Sybase CEP. The third table shows the mapping between Sybase CEP data types and ANSI SQL data types. We recommend that you look in the first two tables for your database server. If your database server is not listed, and if your database server supports ANSI SQL data types, use the third table.

CCL Type	Oracle (Version 10) SQL Type	DB2	MySQL	MySQL MaxDB
BLOB	VARCHAR(N) [6] [7]	VARCHAR(N) [7]	VARCHAR(N) [6] [7]	VARCHAR(N) [7]

CCL Type	Oracle (Version 10) SQL Type	DB2	MySQL	MySQL MaxDB
BOOLEAN	INTEGER	INTEGER	INTEGER	INTEGER
INTEGER	INTEGER	INTEGER	INTEGER	INTEGER
LONG	NUMBER(21)	BIGINT	BIGINT	FIXED(38)
FLOAT	FLOAT	FLOAT	DOUBLE	FLOAT
STRING [3]	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)
INTERVAL	NUMBER(21)	BIGINT	BIGINT	FIXED(38)
TIMESTAMP	TIMESTAMP(6)	TIMESTAMP	DATETIME	TIMESTAMP(6)
XML	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)

CCL Type	SQL Server	Sybase	PostgreSQL
BLOB	VARCHAR(N) [7]	TEXT	VARCHAR(N) [7]
BOOLEAN	INTEGER	INTEGER	INTEGER
INTEGER	INTEGER	INTEGER	INTEGER
LONG	BIGINT	BIGINT	BIGINT
FLOAT	FLOAT	FLOAT	DOUBLE PRECISION
STRING [3]	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)
INTERVAL	BIGINT	BIGINT	BIGINT
TIMESTAMP	DATETIME	DATETIME	TIMESTAMP
XML	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)

The table below shows the mappings between data types and ANSI SQL data types.

CCL Type	ANSI SQL Type	Description
BLOB	[footnote 1]	A sequence of 0 - 4294967295 (2^32 - 1) bytes, each of which may contain any value from 0 - 255. [4][5]

CCL Type	ANSI SQL Type	Description
BOOLEAN	[1]	If the type is stored as a NUMBER or INTEGER, then false = 0 and true = 1
INTEGER	INTEGER	Integer values between -2147483648 and +2147483647 (-2^31 to 2^31 - 1)
LONG	[1]	Integer values from -9223372036854775808 to +9223372036854775807 (-2^63 to +2^63 - 1
FLOAT	FLOAT	For NUMBER(p,s) when s>0
STRING [3]	CHARACTER VARYING (2147483647)	Character strings
INTERVAL	DAY-TIME INTERVAL [2]	Interval of time, specified as days, hours, minutes, seconds, and fractions of a second. For Sybase CEP INTERVAL data type, the precision is in microseconds.
TIMESTAMP	TIMESTAMP	A date and time specified with a precision as fine as 1 microsecond.
XML	CHARACTER VARYING (2147483647)	A string containing valid XML.

Notes:

1. This data type is not part of the ANSI SQL92 specification.
2. Sybase CEP does not support YEAR-MONTH intervals.
3. On some platforms, Sybase CEP strings may be limited to 65,535 bytes.
4. Although internally Sybase CEP supports BLOBs up to 4GB, the practical maximum size is often 2GB due to other limits. For example, Sybase CEP Server cannot read a file (or write a file) larger than 2GB, so you cannot read a 4GB blob from, or write a 4GB blob to a file.
5. As a practical matter, the larger the amount of data to be handled, the fewer rows per second the server can handle. Large values of type BLOB or of type STRING or type XML will severely limit throughput.

6. Oracle BLOB or MySQL BLOB are not supported for use with Sybase CEP BLOB.
7. When Sybase CEP BLOB data is stored in VARCHAR or another ASCII format (as opposed to a binary format), the BLOB is converted to a string by using bas64 encoding, which converts three bytes of BLOB data to four bytes of ASCII data. This expands the size of the data by 4/3 (which typically means that if the database server's data size limit is 2GB for VARCHAR, then only 1.5 GB of BLOB data can be stored in it). For a little more information about base64 encoding, see *Reading and Writing BLOBs on External Database Servers* on page 60.

Below is a sample CCL code that creates a table that you can write data to on an Oracle database server.

```
-- PURPOSE:
--    This creates a table on an Oracle database server and
--    loads data into that table.
-- ASSUMPTIONS:
--    The database must already have been created and the user
--    must already have permissions to access the database.
-- Create the table.
EXECUTE STATEMENT DATABASE "LocalOracleDB"
[[create table BigData1 (
-- We can't map Sybase CEP INTERVAL to Oracle INTERVAL
-- directly, so we'll store it as a 21-digit integer.
interval_col NUMBER(21),
-- We can't map Sybase CEP BLOB to Oracle BLOB/CLOB
-- directly, so we'll store it as a VARCHAR.
blob_col VARCHAR(1000)
)]]
SELECT SetupStream.command AS command
FROM SetupStream
WHERE command = "CREATE TABLE"
;
-- Load data into the Oracle table.
EXECUTE STATEMENT DATABASE "LocalOracleDB"
[[INSERT INTO BigData1 (interval_col, blob_col)
VALUES (?interval_col, ?blob_col)
]]
SELECT
-- Can't directly map Sybase CEP INTERVAL to Sybase CEP INTERVAL!
TO_LONG(Interval1) as interval_col,
-- Remember that the blob_col is actually VARCHAR on Oracle.
Blob1 AS blob_col
FROM InputFromCSVFile
;
-- To read back the data from Oracle to Sybase CEP,
-- we could use the Sybase CEP Read From Database adapter,
-- which can be hooked up directly to a Sybase CEP input stream.
-- The schema of the Sybase CEP input stream would be required
-- to match the schema of the Oracle table from which we
-- would read.
```

Troubleshooting

This chapter helps you identify and resolve problems that are commonly experienced by users of the Sybase CEP SDKs.

General Tips

Re-compile in-process adapters, UDFs, and server plugins when you upgrade Sybase CEP versions.

Code that runs inside the server (such as in-process adapters, UDFs and server plugins) is not necessarily binary compatible from version to version. If you have written your own in-process code, then each time you upgrade your Sybase CEP Engine, re-compile that code and copy it to the appropriate location, such as the server bin directory.

For out-of-process code, on the other hand, you must re-compile only if the API has changed in a way that is not backwards compatible. Such changes are listed in the `SybaseCEPReleaseNotes.txt` file in the Server doc directory. If you skip over intermediate versions, for example, you jump from version 4.4 to version 5.1, then you may need to read the release notes for each of the intermediate versions to learn about changes that are not backwards compatible.

Upgraded versions of Microsoft Visual Studio .NET if appropriate: When Sybase CEP Engine was upgraded from Version 5.0 to Version 5.1, it required users to upgrade from Microsoft Visual Studio .NET 2003 to Microsoft Visual Studio .NET 2005.

Note: Visual Studio .NET 2005 updates project-related files (for example, .vcproj or .sln), and earlier versions of Visual Studio cannot read the updated files. Therefore, upgrading is irreversible. You must re-create the project-related files in order to downgrade.

Errors When Compiling C-language Adapters, UDFs and RPC Plugins

This section describes common errors you may experience when compiling C-language components.

Problem

If you are using MS Visual Studio and get the following error when you try to Build/Rebuild the project or link the .DLL file:

```
LNK2005: _DllMain@12 already defined in MSVCRTD.lib(dllmain.obj)
```

delete the .cpp file from the project and then add it back to the project.

To delete the file from the project, first open the solution explorer window by going to the menu and clicking on "View", then on "Solution Explorer"). The Solution Explorer window will open on the right-hand side of the MS Visual C IDE window. Click on the .cpp file (for example, MyUDF.cpp) and press the delete key.

To add the file back to the project, go to the menu and click on the menu item "Project" and then "Add Existing Item", and then specify the .cpp file.

Problem
Compile/link error "Error LNK2019: unresolved external symbol __imp__xmlFree"

You need to include the library libxml2 in the list of dependencies. See *Compiling an RPC Plugin* on page 268.

Error Messages When Compiling Java-Language Adapters

This section describes common errors you may experience when compiling Java-language components.

Problem
When you compile, you see one or more error messages similar to the following:

```
MapPublisher1.java:1: package sybase.com.c8.toolbox does not exist
import com.sybase.c8.toolbox.Toolbox;
MapPublisher1.java:33: cannot find symbol
symbol : variable Publisher
location: class MapPublisher1
publisher.publishRow(data);
```

The most likely cause is that your CLASSPATH environment variable is not set properly.

Make sure that you include the following files in your CLASSPATH:

```
C:\Program Files\SybaseC8\Server\adapters\sybasec8-jms-adapters.jar
C:\Program Files\SybaseC8\Server\sdk\java5\sybasec8-sdk-java.jar
```

Errors When Starting the Server

This section describes common errors you may experience when starting Sybase CEP Server.

Problem
When Sybase CEP Server is attempting to start, it exits without apparent reason.

This means there may be another server running that is using the same port number.

Problem
When the Sybase CEP Server is attempting to start, it freezes up after displaying the message "Reading Adapter ADL definitions from..."

This problem has been observed by a customer who was using cygwin (a linux look-alike product) under Microsoft Windows, and who compiled an in-process adapter using the "cygwin" libraries.

If you are running on Microsoft Windows and using cygwin or a similar product under Microsoft Windows, make sure that you compile and link in-process adapters and UDFs with a Microsoft Windows development system such as Microsoft Visual Studio, rather than with the C compiler supplied with cygwin. For out-of-process adapters and other code that is not run as part of the server itself, you may be able to use cygwin development tools.

Error Messages Displayed During Execution

This section describes common errors you may experience during execution.

Problem
Sybase CEP Studio produces an error similar to:

```
In workspace 'Default', top module 'PassThrough':
Error: Cannot find user_output_c8adapter_initialize in library
sdk_adapter_lib.
Error C8_SERVER-3601: Module 'Default/PassThrough/
PassThrough_SDKOutputTutorialCsvFileAdapter' execution failed.
```

This means that Sybase CEP Server cannot find the entry point in the adapter library. This error also is printed in the Sybase CEP Server log. Please review the following questions to see if any of them might be causing your problem.

- Has the library been copied to the bin subdirectory of the Sybase CEP Server directory?
- Is the entry point name spelled correctly? These names are case sensitive.
- Are you debugging with Visual Studio? If so, the debug and release versions may be in different directories.
- Have you copied the adapter files to the correct plugins directories?
- Have you copied the adapter library files to the correct bin directory?

  Note also that the API is in C, not C++, so make sure that the user routines have been properly externed.

Problem
You get an error message similar to one of the following:

- Error: Function or library not found (function_name='my_initialize_func', library_name='my_managed_adapter_lib')').

- Error: Cannot find my_initialize_func in library MyManagedInputAdapter.
- Error: Failed to start program (program_path=
  'C:/Documents and Settings/jsmith/My Documents/SybaseC8Repository/5.2.0/ccx/Default_SDKDemo.ccx',
  reason='Request processing failed: Server returned: Exception Error C8_SERVER-4404: Could not execute register command on container 'http://localhost:6789/Container'. Request processing failed: Server returned: Sender Invalid Request: Error parsing XML: Error: Function or library not found. Library: avgbool Function: avgbool.Error C8_SERVER-4101: Could not execute register command for workspace 'Default'.', server='http://localhost:6789/', workspace_name='Default')
  Errors: 1----- ERROR: Module 'SDKDemo' in workspace 'Default' was not started (05/09/06 09:27:29) -----

Possible causes include:

- Your .adl file might not specify the same library name and function name as the .dll/.so file use.
- The case (for example, upper case vs. lower case) for the C function in the C-language source file might not match the case of the name in the .adl file.
- You may not have copied the .dll/.so file to the correct directory. Copy it to the bin directory of the server, for example,
  `C:\Program Files\SybaseC8\Server\bin`
  or
  `/home/<userID>/sybasec8/server/bin`
- Your .udf file might not specify the same library name and function name as what the .dll/.so file uses.
- The "case" (for example, upper case vs. lower case) for the C function in the C-language source file might not match the case of the name in the UDF file. In your UDF file, look for a line similar to:

  ```
  Function Name="weightedAvg3"
  ```

  and make sure that the name there matches the name of the function in the C source code.
- You may have omitted the "extern C" section of your C file, which specifies that the function names should be externally visible. You should have a section similar to the following:

  ```
  extern "C" {
      USER_FUNCTION_EXPORT void my_func(C8Udf* ctx);
      };  // extern "C"
  ```

  If you will be compiling and executing on both Microsoft Windows and other platforms, your section should look more like the following:

  ```
  // Ensure functions are "exported" properly from dll.
      #if defined(_MSC_VER)
  ```

```
#define USER_ADAPTER_EXPORT __declspec( dllexport )
#else // defined(_MSC_VER)
#define USER_ADAPTER_EXPORT
#endif // defined(_MSC_VER)
// forward declarations of callback functions for the
// in-process adapter
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
USER_ADAPTER_EXPORT C8Bool initialize(C8Adapter *adapter_ptr);
USER_ADAPTER_EXPORT C8Bool execute(C8Adapter *adapter_ptr);
USER_ADAPTER_EXPORT void  shutdown(C8Adapter *adapter_ptr);
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

Problem

You are using a UDF that you compiled on Microsoft Windows and you see an error message similar to the following:

```
Error: failed to start program
(program_path='C:/Documents and Settings/<username>/My Documents/
SybaseCEPRepository/5.2.0/UserDefinedAggregators/Xconcat/
xconcat.ccx',
reason='Request processing failed: Server returned: Exception Error
C8_SERVER-4404: Could not execute a command in container 'http://
<nodename>:<port>/Container'.
Request processing failed: Server returned: Client Invalid Request:
Error parsing XML: Error: Can not load library 'xconcat' with path
'C:/Program Files/SybaseC8/Server/bin\xconcat.dll'.
(nspr_error='error 126', system_error='The operation completed
successfully. ').
Error C8_SERVER-4101: Could not execute loader command for workspace
'Default'.', server='http://localhost:<port>/',
workspace_name='Default')
```

Possible causes include:

- You may not have put the .dll file into the server bin directory.
- When you compile your .dll file, you may have turned on debugging. As a result, your .dll may reference the debug version of the C runtime library (for example, `MSVCR8D.dll`) rather than the non-debug version (for example, MSVCR8.dll). If you don't have a copy of the referenced file, you'll get this error.
- You may have compiled your UDF with project settings that indicate that you want your .dll file to use functions in MFC (Microsoft Foundation Classes).

Possible solutions:

- Put the .dll file into the server bin directory.
- Turn off debugging. In the Microsoft Visual C compiler, you can turn off debugging by performing the following steps:

1. Check whether you're using the debug version of the run-time library.
   a. Build->Project->Properties-
   b. Click on "C/C++".
   c. Click on "Code Generation".
   d. For the Runtime library, you should have "Multi-threaded DLL (/MDd)".
      If you see "Multi-threaded Debug DLL (/MDd)", then switch to the non-debug version, which "Multi-threaded DLL (/MDd)"
2. Check the preprocessor settings
   a. Click on "Preprocessor".
   b. In the Preprocessor Definitions section, you will probably see "WIN32;_WINDOWS;_DEBUG;_USRDLL".
   c. Remove "_DEBUG;".

- You may need to modify or re-create your project and specify that you want to create a DLL that does not depend upon the Microsoft Foundation Classes.

Rebuild the .dll, copy it to the server bin directory then stop and restart the server.

Problem

You are using an in-process adapter that you compiled on Microsoft Windows and you see an error message similar to the following:

```
Error: failed to start program
(program_path='C:/Documents and Settings/<username>/My Documents/
SybaseCEPRepository/5.2.0/UserDefinedAggregators/Xconcat/
xconcat.ccx',
reason='Request processing failed: Server returned: Exception Error
C8_SERVER-4404: Could not execute a command in container 'http://
<nodename>:<port>/Container'.
Request processing failed: Server returned: Client Invalid Request:
Error parsing XML: Error: Can not load library 'xconcat' with path
'C:/Program Files/SybaseC8/Server/bin\xconcat.dll'.
(nspr_error='error 126', system_error='The operation completed
successfully. ').
Error C8_SERVER-4101: Could not execute loader command for workspace
'Default'.', server='http://localhost:<port>/',
workspace_name='Default')
```

Possible causes include:

- You may not have put the .dll file into the server bin directory.
- When you compile your .dll file, you may have turned on debugging. As a result, your .dll may reference the debug version of the C runtime library (for example, MSVCR8D.dll) rather than the non-debug version (for example, MSVCR8.dll - note the missing "D" before the period). If you don't have a copy of the referenced file, you'll get this error.

- You may have compiled with project settings that indicate that you want your .dll file to use functions in MFC (Microsoft Foundation Classes).

Possible solutions:

- Put the .dll file into the server bin directory.
- Turn off debugging. In the Microsoft Visual C compiler, you can turn off debugging by performing the following steps:
  1. Check whether you're using the debug version of the run-time library.
     a. Build->Project->Properties-
     b. Click on "C/C++".
     c. Click on "Code Generation".
     d. For the Runtime library, you should have "Multi-threaded DLL (/MDd)".
        If you see "Multi-threaded Debug DLL (/MDd)", then switch to the non-debug version, which is "Multi-threaded DLL (/MDd)"
  2. Check the preprocessor settings
     a. Click on "Preprocessor".
     b. In the Preprocessor Definitions section, you will probably see "WIN32;_WINDOWS;_DEBUG;_USRDLL".
     c. Remove "_DEBUG;".
- You may need to modify or re-create your project and specify that you want to create a DLL that does not depend upon the Microsoft Foundation Classes.

Rebuild the .dll, copy it to the server bin directory; then stop and restart the server. If you are using Microsoft Windows, you must make sure that the `initialize()`, `execute()`, and `shutdown()` functions are exported from your .dll.

Problem

When you run your out-of-process adapter, you see one or more error messages similar to one of the following:

```
Exception in thread "main" java.lang.NoClassDefFoundError:
org/apache/axis/EngineConfiguration
Exception in thread "main" java.lang.NoClassDefFoundError:
javax/xml/rpc/Service
```

The exact name of the missing class may vary.

The most likely cause is that your CLASSPATH environment variable is not set properly. Make sure that you include each of the .jar files in the directory

```
C:\Program Files\SybaseC8\Server\sdk\java5\lib
```

See *Setting Up Your Environment* on page 279, which lists the .jar files that should be in your CLASSPATH.

Problem
When you run the program, you get a message similar to the following:

```
Could not send tuple: java.io.IOException: Could not resolve url
'ccl://localhost:6789/Stream/Default/JavaInputAdapter1/InStream':
Info: Can not resolve uris
```

The exact URL may be different.

The possible causes include the following:

1. The server is not running.
2. The server is running but the query module is not executing.
3. The URL is incorrect.

Make sure that the server is running and the query is running. After you start the server, you should start the components in the following order:

1. The output adapter (if it is an out-of-process adapter).
2. The query module.
3. The input adapter (if it is an out-of-process adapter)

Make sure that the URL is correct. In Sybase CEP Studio, you can see the URL by clicking on the stream and then clicking on the "Properties" tab for that stream.

Problem
`C8OpenStreamConnForReading()` or `C8OpenStreamConnForWriting()` returns a value other than C8_OK.

Possible causes include:

1. The URI is spelled incorrectly. Remember that the URI is case-sensitive.
2. Sybase CEP Server is not running or is inaccessible from the test client machine.
3. The Sybase CEP Studio has not initialized the stream. If Sybase CEP Studio is not running, or if the query module has not been loaded into Sybase CEP Studio, then the stream will not be initialized and the adapter will not be able to connect to it.
4. If a file is specified, the permissions have not been set correctly to allow reading and writing?

Problem
You try to use a TitleRow (in other words, a list of the column names in the stream schema) and parts of the title row are processed as though they were data. Remove the TitleRow from the input to resolve this issue.

Problem
You see a message similar to:

```
error while loading shared libraries: libstlport.so.5.1: cannot open
shared object file: No such file or directory
```

or

```
Can not load library 'library_name' with path 'some_path'.
(Reason='Failure to load dynamic library (-5977,0)')
```

If you are on a Unix-like operating system, and if you are trying to compile and register a query, then you may not have set the LD_LIBRARY_PATH environment variable to include the directory the library file is in.

To solve the problem, update your LD_LIBRARY_PATH and then restart the server. Also check that your PATH is correct.

Problem

You see a message similar to the following when you try to compile a CCL module that uses a UDF:

```
CCLC2036: Error: Unknown operator 'MyFunc( long )'.  MyFunc(x)
```

The error indicates that the compiler did not recognize the function named MyFunc with parameters of the specified types (the type "long" in this example).

Possible causes include:

- You may not have created the .udf file.
- The contents of the .udf file may be incorrect. For example, you might have the wrong values for the Function Name, Library, or CclName, or you might have the wrong data types specified for the input and output parameters. Check that you used the correct values.
- You may not have copied the .udf file to Sybase CEP Studio's plugins directory. (You might have copied the file only to the server's plugins directory.)

HTTP and SOAP Plugin Configuration

The c8-services.xml configuration file contains preferences related to interfacing Sybase CEP with other systems.

- Database Servers:
  - Oracle Database Servers (natively, or via ODBC).
  - Other database servers (for example, MySQL, PostgresSQL, and SQLServer) via ODBC.
- RPC servers.

Sybase CEP Server treats an external data source (or data destination) as a "service". For each service, the `c8-services.xml` file must contain a unique "service name", along with the information required to access that service (for example, username, password, and so on). The Sybase CEP *Installation Guide* describes how to configure your `c8-services.xml` file with the service name and most other information needed to access remote database servers and RPC servers.

The Sybase CEP *Installation Guide* does not describe configuration preferences for the HTTP plugin and the SOAP plugin, either of which can be used to call remote procedures on RPC servers. Those HTTP plugin and SOAP plugin configuration preferences are described here.

The generic HTTP plugin expects the following configuration preferences in the `c8-services.xml` file:

HttpURI	Required	This is the HTTP URI to which the request should be sent. For example: `http://localhost/cgi-bin/getStock-Price.cgi`
HttpTimeout	Optional	Specifies the HTTP connection timeout, either an integer specifying the number of seconds or in the following format: `[D day[s]][ ][HH hour[s]][ ][MM minute[s]][ ][SS[.FF] second[s]]` For example, "3 minutes 12.5 seconds". The default value is 30 seconds.
HttpKeepAlive	Optional	TRUE or FALSE, indicating whether to use keep-alive connections. The default value is FALSE.
HttpEnableRedirects	Optional	TRUE or FALSE, indicating whether to follow redirects. The default value is TRUE.

HttpEnableLogging	Optional	TRUE or FALSE, indicating whether to log arguments and return values for all plugin calls. The default value is FALSE.
HttpEnableDebugging	Optional	TRUE or FALSE, indicating whether to print debug information from the plugin (for example, request/response dumps). The default value is FALSE.
HttpRequestHeaders	Optional	Specifies extra HTTP request headers. Multiple headers must be separated with the 2-character sequence Carriage Return and Line Feed (represented as "\r\n below). For example, **Accept: text/xml\r\n SOAPAction:"TempConvert#f2c"**

The generic SOAP plugin expects the following configuration preferences in the `c8-services.xml` file:

SoapUri	Required	Indicates the URI of the SOAP resource.
SoapMethod	Required	Specifies the name of the SOAP method to call, For example, `GetStockPrice` .
SoapAction	Optional	Indicates the SOAPAction HTTP header. The default value is `<SoapURI>#<SoapMethod>`.
SoapVersion	Optional	Indicates the version of SOAP in use, either 1.1 or 1.2.
SoapMethodNamespace	Optional	Indicates the namespace to use for the method node. Defaults to the same value as SoapUri. If you set this to the empty string, then the method node is written without a namespace.
SoapValuesNamespace	Optional	Indicates the namespace to use for the values node. Defaults to the same value as SoapMethodNamespace. If you set this to the empty string, the values node is written without a namespace.
SoapEncodingStyle	Optional	"Document" (the default) or "Rpc", Specifies the encoding style for the SOAP request and response, either defaults to "document" or "Rpc".
SoapXsdNamespace	Optional	Specifies the xsd namespace URI. The default value is `http://www.w3.org/2001/XMLSchema`. Old SOAP servers may require the old version, which is: `http://www.w3.org/1999/XMLSchema`

SoapXsiNamespace	Optional	Specifies the xsi namespace URI. The default value is `http://www.w3.org/2001/XMLSchema-instance` However, old SOAP servers might require the old version, which is: `http://www.w3.org/1999/XMLSchema-instance`
SoapDotNet20Compat	Optional	TRUE or FALSE, indicating whether .NET3 compatibility mode should be enabled. .NET3 compatibility mode causes timestamps to be generated without time zones. The default value is FALSE.
SoapTimestampFormat	Optional	Specifies the timestamp string serialization format, for example: `YYYY-MM-DDTHH24:MI:SS.FFZTZH:TZM` The default value is the default SOAP format, which requires the "T" before the hour and the "Z" before the time zone information. These extra characters are compatible with the Sybase CEP format; Sybase CEP treats them as literals.)

The remote service entry can be configured for a number of additional settings, as described in the Sybase CEP *Installation Guide* section titled "Setting Optional Preferences for Services".

To learn the syntax of RPC and Database statements and subqueries, refer to Sybase CEP *CCL Reference Guide* .

Stream URIs

Read this section if you plan to use out-of-process adapters (or any other program that uses the Sybase CEP SDKs), distributed queries, high availability, or if you want to make URIs portable across servers.

What Is a URI and What Is It Used For?

Each Sybase CEP stream has a unique stream URI which provides enough information that a program can connect to the stream.

You must also specify a URI when you connect streams in different projects. If you are already familiar with URLs, it may be helpful to know that a URI is a lot like a URL.

The simplest use of stream URIs is with an out-of-process adapter, which runs as a separate process rather than as part of the server. For an out-of-process input adapter to connect to an input stream and write data to that stream, the adapter must know the URI of that stream. Similarly, an out-of-process output adapter must know the URI of the stream from which it reads. Any other program that uses a Sybase CEP SDK to access a stream must also know the URI of the stream.

A more complex use of stream URIs occurs with parallel queries; the adapter must write data to multiple instances of a stream, and thus must know the URI of each of those instances of the stream. This is explained in more detail in the section titled *URIs and Distributed Queries* on page 366.

Another use of stream URIs is with the High Availability (HA) feature. When using HA, Sybase CEP Server cluster has a manager and one or more containers. The manager coordinates the work among the containers. The containers not only run queries; they also run streams and adapters. When a manager or container dies, a different manager or container takes over the work, and so a program may need to re-connect its stream if the stream is now running on a different computer. Thus, Sybase CEP Server must use stream URIs that can be resolved to the current location of the stream, rather than hard-coding a host as part of the URI.

Types of URIs

This section discusses logical vs. physical stream URIs, and absolute vs. relative stream URIs.

Note: SybaseCEP URI naming follows the RFC 3986 standard for URIs. A CCL URI is a URI with the "ccl" scheme.

Logical vs. Physical Stream URIs

Sybase CEP stream URIs may be logical or physical URIs.

User-written programs, such as out-of-process adapters, generally specify a logical URI. The manager resolves this into a physical URI. The physical URI includes information about which container the stream is running on.

- Logical URIs start with "ccl://".
- Physical URIs start with "http://".

Absolute vs. Relative Stream URIs

URIs that start with "http:" or "ccl:" are absolute URIs. You may also specify relative URIs, which are relative to the local Sybase CEP Server manager.

For example, the following binding is valid:

```
/Stream/wkspc2/MyProject/strm_OrderUpdate~1
```

Note the "~1" at the end of the relative URI. This looks up the stream named `strm_OrderUpdate~1` in the current manager. "wkspc2" is the name of the workspace. If the project is running on host "dev7" and port "6782", then the binding

```
/Stream/wkspc2/MyProject/strm_OrderUpdate~1
```

is the same as

```
ccl://dev7:6782/Stream/wkspc2/MyProject/strm_OrderUpdate~1
```

By using relative bindings, you make projects much more portable across servers.

URIs and Distributed Queries

Distributed queries may be pipelined or parallelized, or both.

If you are using the parallel query feature of Sybase CEP Engine, a stream will be split into separate "instances" and the addresses of the stream instances will be the same except for a "~" followed by a number, as shown below:

```
ccl://chili:6789/Stream/StockWS/StockEPSqm/Subq/StockPriceFeed~1
ccl://chili:6789/Stream/StockWS/StockEPSqm/Subq/StockPriceFeed~2
ccl://chili:6789/Stream/StockWS/StockEPSqm/Subq/StockPriceFeed~3
```

Start with the number 1, and increase by 1 for each additional stream you want to distribute the data across.

An input adapter attempting to write to stream instances in a parallel query must resolve each of these logical URIs into a physical URI by passing that individual logical URI to a `resolve_uri()` function. This resolves in a separate physical URI for each stream instance, and your adapter splits the data across all of these physical URIs.

An output adapter may subscribe to the individual stream instances (the ones whose names end with "~#", where the "#" represents a positive integer) or the combined stream. If the output adapter specifies the URI without the "~#", the adapter will get the combined output of the stream. If the output adapter specifies the URI with the "~#", then of course the adapter will get the output of just one individual stream instance. Similarly, if an input adapter publishes a row to the stream URI with a "~#", then just that stream instance gets the row; if the adapter publishes to a stream URI without a "~#", then each instance of the stream will get a copy of the row.

If you set the number of project instances to 1, then there will only be one stream instance, and accessing a stream with a name ending with "~1" will not work.

If you are pipelining queries across projects (in other words, sending data from an output stream in one project to an input stream in another project), you need to specify the URIs of the streams when you bind a stream in one project to a stream in another project. For information about binding streams from inside Sybase CEP Studio, see the Sybase CEP *Studio Guide* .

URIs and High Availability

A description of the URI IP address, its typical behaviour, and how URIs behave in high availability systems.

The physical URI of a stream includes the name or IP address of the computer on which the stream is running. Anything that might cause the stream to run on a different computer will cause the stream's physical URI to change. The CCL URI will not change, however, so we recommend that you use the CCL URI rather than the HTTP URI whenever possible.

For example, in an HA (High Availability) system, if a server dies, the streams on that server will be moved to another server. If you are using the High Availability feature and the Parallel Query feature together, then if one of these streams to the parallel query moves, your out-of-process adapter will need to get the stream's new physical URI by calling the resolve_uri() function again.

Finding the URI of a stream

The easiest way to find the URI for a particular stream is to use Sybase CEP Studio.

Follow the steps below to find a stream's URI in Sybase CEP Studio.

1. Start Sybase CEP Studio

2. Load the query module that contains the stream (if you have not already loaded it).
3. Click on the stream in the Explorer View, which is usually the upper left-hand pane of Sybase CEP Studio. This displays the "Properties" View for the adapter.
4. Read the Stream URI and the Http URI from the Properties View.

Summary

This section provides short descriptions for out-of-process adapters, distributed queries, and high availability.

Out-of-process adapters: Out-of-process adapters (and other programs that access Sybase CEP Server through a Sybase CEP SDK) that are used without Parallel Queries should use the logical URI, which can be copied from a stream viewer window or can be composed dynamically based on the name of the stream, and so on.

Distributed Queries: Distributed queries may be pipelined or parallelized (or both). When using out-of-process adapters with parallel queries, if you need all the physical URIs for all of the streams, then compose the logical URIs and then call resolve_uri() to get the physical URIs for each of the streams.

High Availability: Programs that need to contact an HA container should store the logical URI, resolve it to a physical URI, and then connect to that physical URI. If a container dies and its work (including streams) fails over to another container, then re-resolve the logical URI to get the new physical URI, and then connect to the new physical URI.

Warning! If a manager is lost because its entire host computer shut down and a computer with a passive manager is standing by, the standby computer should be a full mirror of the original host computer. The reason for this is that the standby communicates with the DDNS process and takes the hostname of the original computer. This means that if there are other (non-Sybase CEP) processes running on the old host that are accessed from other computers, anyone trying to contact those processes sees the replacement host under the old hostname and, but the other processes will be missing.

Connecting to Streams over a Network

This section describes native stream formats.

As we mentioned earlier, the Sybase CEP Engine reads and writes data only in its native stream formats. Every adapter writes to (or reads from) a stream in one of these formats. The formats are:

- Binary
- XML
- CSV

This chapter describes how to connect directly to a Sybase CEP stream over the network and how to publish or subscribe to a stream after you've connected to it directly.

Data Stream URI

Each native data stream has a URI that uniquely identifies the stream.

The stream URI can be used to subscribe or publish to a stream. The stream URI contains information about the Sybase CEP Server hostname and port number and Stream URI "path" on this server. For example, the following stream URI points to stream /Stream/Default/PassThrough/InTrades running on Sybase CEP Server located on localhost and port 6789:

```
ccl://localhost:6789/Stream/Default/PassThrough/InTrades
```

Note that the URI uses "ccl:" instead of "http:".

Subscribing to a Data Stream

To receive data from a Sybase CEP data stream, the subscriber must perform an HTTP GET request to the Sybase CEP Server.

The HTTP GET request must include the following HTTP headers and an empty HTTP body:

```
GET <path> HTTP/1.0\r\n
Host: <server-hostname-or-ip>\r\n
X-C8-StreamFormat: <stream-format>\r\n
X-C8-StreamFormatOptions: <stream-format-options>\r\n
\r\n
```

where

- <path> is the path portion of the Stream URI.

- <server-hostname-or-ip> is the hostname portion of the Stream URI.
- <stream-format> is the data stream format (CSV or XML); if the format is omitted, then binary format is assumed.
- <stream-format-options> is the format specific options. (If the stream-format-options are omitted, the default values for format options are used.)

The "\r\n" represents the 2-character sequence carriage return and line feed, not the 4-character literal "\r\n".

Note:

Note: The header is followed by a blank line terminated with a carriage return and line feed.

The client should expect to receive an HTTP response tha contains an HTTP header and an HTTP body. Since the data stream may continue indefinitely, the HTTP body is essentially an endless series of messages, each of which will have data in the stream format specified in the HTTP request. A typical HTTP response looks similar to the following:

```
HTTP/1.0 200 OK\r\n<header1>: <header-value1>\r\n<header2>: <header-
value2>\r\n
...
<headerN>: <header-valueN>\r\n
\r\n
<data-in-specified-data-stream-format>
```

where

- <header1>:<header-value1>, ... ,<headerN>:<header-valueN> are the regular HTTP headers which should be ignored by the subscriber.

As before, the "\r\n" represents the two character sequence of a carriage return followed by a line feed.

The Sybase CEP HTTP response is different from most HTTP responses from entities other than Sybase CEP Server. The output data stream is a series of messages that may continue indefinitely, so there is no way for the response to specify the length of the message in the header. Furthermore, the response is a series of messages and each of those messages should be dealt with when the message arrives. The recipient does not wait until the end of the HTTP response to start processing the messages in it.

For example, the following HTTP request subscribes to stream `/Stream/Default/Finance/StreamIn` :

```
GET /Stream/Default/Finance/StreamIn HTTP/1.0\r\n
Host: localhost\r\n
X-C8-StreamFormat: CSV\r\n
X-C8-StreamFormatOptions: TitleRow=true,
TimestampColumn=true,
TimestampColumnFormat=YYYY/MM/DD HH24:MI:SS\r\n
\r\n
```

Note: Due to limitations on the page width of this manual, the X-C8-StreamFormatOptions line is shown as split across multiple lines, but is really a single line. Note carefully where the "\r\n" sequences are shown.

The subscriber receives trading data in CSV format in the following HTTP response:

```
HTTP/1.0 200 OK\r\n
Date: Wed, 28 Jan 2005 10:23:53 GMT\r\n
\r\n
Timestamp,Symbol,Price,Volume\r\n
"2005/01/28 10:23:54",ABC,11.40,300000\r\n
"2005/01/28 10:23:55",XYZ,32.84,1260000\r\n
...
```

Publishing to a Data Stream

To publish data to a Sybase CEP data stream, perform an HTTP POST request to the Sybase CEP Server.

The HTTP POST request must include the following HTTP header and an infinite HTTP body with data in the specified data stream format:

```
POST <path> HTTP/1.0\r\n
Host: <server-hostname-or-ip>\r\n
X-C8-StreamFormat: <stream-format>\r\n
X-C8-StreamFormatOptions: <stream-format-options>\r\n
\r\n
<data-in-specfied-data-stream-format>
```

where

- <path> is the path portion of the Stream URI.
- <server-hostname-or-ip> is the hostname portion of the Stream URI.
- <stream-format> is the data stream format (CSV or XML).
- <stream-format-options> is the format specific options. If the stream-format-options are omitted, the format defaults to binary.

The publisher should expect no response on the HTTP request it sends. When done publishing data, the publisher should close the TCP connection.

For example, the following HTTP request publishes trading information in CSV format to `/Stream/Default/Finance/StreamIn`

```
POST /Stream/Default/Finance/StreamIn HTTP/1.0\r\n
Host: localhost\r\n
X-C8-StreamFormat: CSV\r\n
X-C8-StreamFormatOptions: TitleRow=true,TimestampColumn=true,
  TimestampColumnFormat=YYYY/MM/DD HH24:MI:SS\r\n
\r\n
Timestamp,Symbol,Price,Volume\r\n
```

```
"2005/01/28 10:23:54",ABC,11.40,300000\r\n
"2005/01/28 10:23:55",XYZ,32.84,1260000\r\n
"2005/01/28 10:24:06",XYZ,32.74,6300000\r\n
"2005/01/28 10:24:32",ABC,12.01,50000\r\n
...
```

Note: Due to formatting limitations, the X-C8-StreamFormatOptions line is shown as split across multiple lines, but is really a single line. Note carefully where the "\r\n" sequences are shown.

Infiniband Support

If you are using a pure Infiniband network, you can configure Sybase CEP to send messages between servers over the SDP, providing high performance, low latency communication.

To use SDP:

- Set the EnableAutoSDP flag in the **c8-server.conf** file to true.
- Set the Hostname to a valid Infiniband address.
- Set the ListenTo port to an Infiniband port.

If the EnableAutoSDP flag is set to false, or if either the host or listening port are not set to valid Infiniband values, Sybase CEP uses standard TCP/IP communication.

```
<!-- Sybase CEP Server -->
    <section name="C8/Server">
        <section name="Common">
            <!-- the server's hostname -->
            <preference name="Hostname">CONFIG_HOST</preference>
           <!-- the server's hostname or IP that it would listen to
-->
            <preference name="ListenTo">0.0.0.0</preference>
           <!-- the server's port number that it would listen to -->
            … …
          <!-- this flag used for enabling data exchanging over
Infiniband network via SDP -->
         <!-- automatically while hostname, ListenTo and ManagerUri
are both set to Infiniband address. -->
            <preference name="EnableAutoSDP">true</preference>
            … …
```

Note: Infiniband support is only available on Linux and Solaris.

Data Stream Formats

A description of Binary, Comma-Separated Value (CSV), and XML data types in Sybase CEP.

Binary Data Stream Format

Binary Data Stream Format is the primary data stream format for Sybase CEP Engine. It is very fast and efficient.

CSV Data Stream Format

Sybase CEP Engine can use CSV Data Stream Format, which is compatible with the CSV data files format used by Microsoft Excel and many other applications.

The box below shows the valid syntax for the Data Stream. The syntax descriptions include characters (such as square brackets) that are not literals in this context but instead have special meanings. These special meanings are explained below:

- Square Brackets:
  Square brackets ("[]") indicate optional items.
  For example, in the following line:
  [too] bar
  the "too" is optional, while the "bar" is required. If the "too" occurs, it must occur only once.
- Ellipsis:
  An ellipsis ("...") after an element means that the element may be repeated one or more times. For example, in the following line:
  PLAIN_CHAR ...
  the ellipsis means that after the first PLAIN_CHAR you may have additional PLAIN_CHARs.
- Capitalized Words:
  To avoid confusion with punctuation marks when a punctuation mark is part of what you must type, the punctuation mark is written as a capitalized word. For example, "COMMA ColumnName" means to put the comma character (",") followed by a column name. Similarly, QUOTE means to put one double quote character (" ' ").

Putting all this together, the following line

Tuple = [Field [, Field ...]] NEWLINE

means that a tuple can have 0 fields, 1 field, or more fields, and if there is more than 1 field then the fields must be separated by commas. The Tuple must be terminated with a NEWLINE. For example, all of the following are valid values for Tuple:

col1

col1, col2

col1, col2, col3

Each of these would of course be terminated with a NEWLINE. An empty line terminated with a NEWLINE would also be valid.

```
Stream = [ Headers ] [ Tuple ]
Tuple = [ Field [ , Field ... ] ] NEWLINE

Note: Fields correspond to tuple field values.
NonEmptyTuple = Field [ , Field ... ]
Headers = <same as Tuple, but fields correspond to tuple field
names.>
Field = [QUOTE] [ PLAIN_CHAR ...] [QUOTE] |
  " [ (ANY_NONQT_CHAR | DOUBLEQUOTE) ... ] "
QUOTE = single quote (ascii 0x22) or double quote ascii 0x27).
Note that if you start a quoted string with a particular type
of quote, you must close with the same type of quote.  For example:
these are invalid:
    'Hi"
    "Bye'
DOUBLEQUOTE = '""' representing a single '"' within the field value.
ANY_NONQT_CHAR = <any ASCII character except for '"' (0x22) and
    including line separators>
PLAIN_CHAR = <any printable ASCII char except ',' and '"',
   i.e. ASCII 0x21 and ASCII 0x23 - 0x7e>
NEWLINE = CR  |  LF  |  CR LF
CR = the Carriage Return character (0x0D)
LF = the Line Feed character (0x0A)
```

By default, Sybase CEP reads and writes the NEWLINE using the convention for the current platform, for example, CR LF for Microsoft Windows and LF for UNIX-like operating systems.

A LineEndCRLF format option may be used to specify the line ending. If the value of this option is 'true', CRLF line ending is used; if it is 'false', LF ending is used.

For example, the CSV trades stream can look like the following:

```
Timestamp,Symbol,Price,Volume\r\n
"2005/01/28 10:23:54",ABC,11.40,300000\r\n
"2005/01/28 10:23:55",XYZ,32.84,1260000\r\n"
"2005/01/28 10:24:06",XYZ,32.74,6300000\r\n
"2005/01/28 10:24:32",ABC,12.01,50000\r\n
...
```

The CSV Data Stream format has the following options:

- TitleRow (true or false) specifies whether the title row is present or not.
- TimestampColumn (true or false) specifies whether the first column is the message's timestamp column or not.

- TimestampColumnFormat (string) specifies the format of the TimestampColumn if it exists. If the format is not specified, then the row timestamp will be a 64-bit integer number whose value is the number of microseconds since midnight January 1, 1970 GMT.

Note that for input adapters:

- If you do not use a title row, then the number of fields and the order of fields are assumed to be exactly the same as the number and order specified in the stream's schema.
- If you do use a title row, then
  - The column names in the title row must exactly match the column names specified in the stream's schema.
  - The columns may appear in any order, as long as the order of the data values is the same as the order of the column names in the title row. The only exception is that if you use a row timestamp, then the row timestamp must always be the first column, and the name of that column does not matter.
  - If there are "extra" columns in the input, those are ignored; only the columns whose names match are used.

For output adapters:

- The order of the columns is the same as the order specified by the schema. There is no way to change this. If you specify that the row timestamp should be included in the output, then the row timestamp is always the first column.)
- If you specify that the output should include a title row, the names of the columns in that title row are exactly the same as the names in the schema.

XML Data Stream Format

Sybase CEP Engine can use XML Data Stream Format, which is compatible with the XML data format used by many third party applications.

The XML data stream consists of an infinite number of Tuple elements following one after another. There is no single root element in the stream. Each Tuple element must conform to the following schema:

```
<xs:schema xmlns="http://www.sybase.com/cpx/2004/03/"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.sybase.com/cpx/2004/03/"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
<xs:element name="Tuple">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Field" maxOccurs="unbounded">
    <xs:complexType>
     <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="Name" type="xs:Name"
                      use="required"/>
      </xs:extension>
```

```
    </xs:complexContent>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="Ts" type="xs:long" use="optional"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

Note: Tuple fields' names must exactly match the column names specified in the stream's schema.

The "Ts" attribute in the "Tuple" element contains the tuple's timestamp as a number of microseconds since 00:00:00 January 1, 1970.

For example, the XML trades stream can look as follows:

```
<Tuple Ts="11032230450" xmlns="http://www.sybase.com/cpx/2004/03/">
xmlns="http://www.sybase.com/cpx/2004/03/">
  <Field Name="Timestamp">2005/01/28 10:23:54</Field>
  <Field Name="Symbol">ABC</Field>
  <Field Name="Price">11.40</Field>
  <Field Name="Volume">300000</Field>
</Tuple>
<Tuple Ts="11032230454" xmlns="http://www.sybase.com/cpx/2004/03/">
  <Field Name="Timestamp"> 2005/01/28 10:23:55</Field>
  <Field Name="Symbol">XYZ</Field>
  <Field Name="Price">32.84</Field>
  <Field Name="Volume">1260000</Field>
</Tuple>
<Tuple Ts="11032230464" xmlns="http://www.sybase.com/cpx/2004/03/">
  <Field Name="Timestamp"> 2005/01/28 10:24:06</Field>
  <Field Name="Symbol">XYZ</Field>
  <Field Name="Price">32.74</Field>
  <Field Name="Volume">6300000</Field>
</Tuple>
<Tuple Ts="11032230467" xmlns="http://www.sybase.com/cpx/2004/03/">
  <Field Name="Timestamp">2005/01/28 10:24:32</Field>
  <Field Name="Symbol">ABC</Field>
  <Field Name="Price">12.01</Field>
  <Field Name="Volume">50000</Field>
</Tuple>
...
```

The XML Data Stream format has no options.

Sybase CEP Adapters

This section describes the input adapters and output adapters that Sybase CEP provides.

Some of these adapters are in-process adapters and some are out-of-process adapters. If an adapter is an out-of-process adapter, it is not started automatically by the server, and the description of the adapter will include instructions on starting the adapter.

Configuring Adapters

Most adapters supplied by Sybase CEP have properties that the user can modify.

Some of these properties have default values and some do not. Set the properties to values that are appropriate for your data.

You can set adapter properties using the CCL Attach Adapter statement or through the user interface of . See the Sybase CEP *CCL Reference Guide* and the for more information.

Each adapter and its properties are described in more detail later in this chapter. Some adapter properties may interact with server parameters that are set in the server configuration file, named `c8-server.conf`.

Setting the Base Folder for File Input/Output Adapters

Specify a directory as the adapters base folder when you install Sybase CEP Server. You can change the base folder by editing the configuration file and modifying the value for the "C8/Adapters/ReadWriteBaseFolder" preference. The adapters base folder tells Sybase CEP Server where to find the data files that an input or output adapter reads or writes. The data files for such adapters must be stored somewhere under the adapters base folder on . This prevents Sybase CEP Server from accidentally interfering with files used by other software.

Adapters supplied by Sybase that access data files include:

- ReadFromCSVFile
- WriteToCSVFile
- ReadFromXMLFile
- WriteToXMLFile
- ReadFromRegularExpressionFile

Relative Paths

In addition to restricting where data files can be stored, the adapters base folder allows you to specify relative paths to data files in your projects. Most people build their projects on a different computer than the one where the project will ultimately be deployed. Because the directory structure on the two computers may be entirely different (they may be running

different operating systems, for example), using relative paths for data files allows you to deploy your project without having to modify the file name adapter properties.

When you attach a file adapter to a data stream, you must specify the path to and name of the file that the adapter reads from or writes to. You start the path with one of two variables: $BaseFolder or $ProjectFolder. Sybase CEP Server replaces $BaseFolder with the value of the "C8/Adapters/ReadWriteBaseFolder" preference in the server configuration file. It replaces $ProjectFolder by linking $BaseFolder with the path to your project .cpp file on the client, relative to the SybaseC8Repository. The repository is the directory uses for its preferences file, environment file, security file, the examples directory, and its temporary directory.

If you use the Sybase CEP Eclipse Plug-In, the workspace directory is equivalent to the repository directory. By default, if you install and Sybase CEP Studio on the same computer, the repository and the adapters base folder are set to the same directory. If you create your project in a directory outside of the repository, replaces $ProjectFolder with the absolute path to the project directory without inserting $BaseFolder.

Example

If the repository on your client Windows computer is set to

```
C:\Documents and Settings\jsmith\My Documents
\SybaseC8Repository\3.0
```

and you create a project under the repository in the directory MyProjects\Project1, then the full path to your project on the client computer is

```
C:\Documents and Settings\jsmith\My Documents
\SybaseC8Repository\3.0\MyProjects\Project1
```

Put your adapter's data file in a subdirectory named Data. When you attach an input adapter to one of your data streams, you specify the data file name as $ProjectFolder/Data/MyData.csv.

On your Linux Sybase CEP Server, set your adapters base folder to /var/C8/BaseFolder.

When you compile and run your project on , it looks for the data file with the following path:

```
/var/C8/BaseFolder/MyProjects/Project1/Data/MyData.csv
```

This is the adapters base folder on the server (/var/C8/BaseFolder) plus the relative path of the project (/MyProjects/Project1) plus the rest of the path and file name specified in the project (/Data/MyData.csv).

Now say that you create a project outside of the repository in this directory:

```
C:\Documents and Settings\jsmith\My Documents\MyProjects
\Project1
```

In this situation Sybase CEP Server would replace $ProjectFolder/Data/MyData.csv with

```
/Documents and Settings/jsmith/My Documents/MyProjects/
Project1/Data/MyData.csv
```

If you instead specify `$BaseFolder/Data/MyData.csv` as the file name, regardless of where the project is stored on the client computer, would look for the data file with this path:

```
/var/C8/BaseFolder/Data/MyData.csv
```

This is the adapter's base folder followed by the rest of the path and file name specified in the project.

Reading and Writing BLOB Data

This section describes reading and writing BLOB data with adapters.

The in-process adapters supplied by Sybase CEP handle the BLOB data type. When data is read or written in CSV format (for example, via the ReadFromCSVFile adapter or the WriteToCSVFile adapter), each BLOB is represented as a string of hexadecimal digits in which each byte of BLOB data is represented by 2 hexadecimal digits and thus 2 bytes. Because CSV files themselves are limited to 2GB (Sybase CEP cannot read or write files larger than 2GB), the effective limit on the size of a BLOB in a CSV file is 1,073,741,823 bytes, or 1GB.

Setting Up the Environment for Java Adapters

This section provides a list of Java adapters you can use with Sybase CEP.

Adapters that are written in the Java Programming language include:

- JDBC Input Adapter
- JDBC Output Adapter
- JMS Input Adapter
- JMS Output Adapter
- JMS Reliable Adapter (Guaranteed Delivery)
- Java Email Output Adapter

These adapters must have the environment, including the CLASSPATH environment variable, set properly. For more information, see *Setting Up Your Environment* on page 279. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Sybase CEP or written by you.

Setting Up the Environment for the JMS Adapter

This section describes how to configure your system to use the Sybase CEP JMS adapters. Sybase CEP provides a JMS input adapter and a JMS output adapter. Each of these adapters is effectively a client in a JMS system.

Before you can use the JMS adapters, you will have to install and configure some software, including a JMS server. The Sybase CEP JMS adapters are is designed to work with any JMS Server. To make our discussion more concrete, we will sometimes refer to the BEA™ WebLogic™ JMS server, but you can use a different JMS server.

Note: This section of the manual focuses on using a SybaseCEP JMS Output adapter, but most of the steps are the same for a Sybase CEP JMS Input adapter.

JMS Adapter prerequisites

Lists the prerequisites for the JMS adapter.

You will need the following:

- Sybase CEP Server and Sybase CEP Studio.
- A JMS Server, such as the BEA™ WebLogic™ server or the Joram distribution. Other JMS servers may also be used.
- The Java JDK 1.4.2 or later. Not all JDKs are compatible. Only the Sun™ JDK is known to be compatible.

Configuring and Setting Up Your JMS Server

This section describes how to configure and set up your JMS server.

Create and Deploy a New JMS Server

The instructions for this depend upon which JMS server you are using, and are not included in this Sybase CEP document.

Please see the instructions for your JMS server.

Create a New Connection Factory for the Server, and Deploy It.

Connection factories are objects that enable JMS clients to create JMS connections. You will need to create a Connection Factory before the Sybase CEP JMS Adapter which is a JMS client can connect to the JMS server. The exact instructions for doing this depends on which JMS server you are using.

Set the Factory Name and JNDI Name for your new Connection Factory. The JNDI Name is the name you give the Sybase CEP JMS adapters so they can look up this JMS Connection Factory.

For example:

Name: cgTopic

JNDI Name: weblogic.jws.jms.TopicConnectionFactory

After you create the Connection Factory, you must deploy it.

If necessary, create a JMS Module to hold your Connection Factory. Specify the Connection Factory name and JNDI name where necessary.

If necessary, create a Topic and give it the same name as the JNDI name.

Set Up a Destination Topic for the JMS Server.

Configure a new JMS Topic for your JMS server. Set the Topic Name and the JNDI Name for that topic. The JNDI Name is the name you will give to lookup the JMS Topic.

Example:

Name: MyJMSTopic

JNDI Name: weblogic.jws.jms.MyJMSTopic

If explicit deployment of this topic is required, then deploy the topic.

Note: Some JMS Servers can have destination queues and/or topics. Sybase CEPSybase CEP publish/subscribes only to topics, not queues.

Configuring and Setting Up Sybase CEP

This section describes how to configure and set up Sybase CEP to work with the JMS adapters.

Sybase CEP Studio

Using Sybase CEP Studio, create a simple CCL module to test the unmanaged JMS Output Adapter.

In Studio, create an input stream and attach an input adapter to that stream. You can use whatever data source you wish. Using the JMS Output Adapter, we subscribe to this stream and publish the stream data to the desired JMS Topic destination.

Sybase CEP JAR files

In order to work with JMS adapters, you need theSybase CEP .jar files that are installed on your system automatically when you install Sybase CEP Server.

Testing the Sybase CEP JMS Adapter

Provides directions on testing the Sybase CEP JMS Adapter.
Make sure that you have all of the required .jar files:

- c8-jms-adapters.jar
- c8-sdk-java.jar

- jaxrpc.jar
- jms.jar
- axis.jar
- commons-logging-1.0.4.jar
- commons-discovery-0.2.jar
- saaj.jar
- wsdl4j-1.5.1.jar
- activation.jar
- mail.jar
- castor-0.9.9.jar
- xercesImpl.jar

You typically need one or more .jar files provided with your JMS server. For example, if you are using the BEA WebLogic JMS server, then you will need weblogic.jar, which is supplied by BEA.

The location of the .jar file for your JMS server depends on the web server you are using. For example, `weblogic.jar` is found in your `BEA\weblogic81\server\lib` directory.

Although most of the required .jar files already reside in C:\Program Files\SybaseC8\Server \sdk\java5\lib, a few are not found there. The exceptions are

`c8-jms-adapters.jar` and `c8-sdk-java.jar`, which are found in the SybaseC8\Server\sdk\java directory.

Make sure that all the required processes are up and running:

a) Make sure that your JMS server is running.
b) Make sure that the Sybase CEP Server is started. To start the server, go to Start -> All Programs->Sybase CEP ->Server->Sybase CEP Server
c) Make sure that your CCL Module is running. Do this by starting the Sybase CEP Studio and then running your CCL module from within Sybase CEP.
d) Start the JMSAdapter, so that messages from Sybase CEP get sent to the desired JMS Server topic destination.
e) Set your CLASSPATH so that it includes all the necessary jar files.
f) Give the command to start the JMSAdapter. The command below is appropriate for the WebLogic JMS server. If you are using a different server, you will have to change some items, including the boldfaced items and probably the port. The following is a single command; we have spread it across multiple lines to make it easier to read.

```
java  -Dc8.baseHostPort=localhost:6789
        com/sybasec8/adapter/JMSAdapter
        --messageType=TextMessage
        --topic=
weblogic.jws.jms.MyJMSTopic

        --factoryName=
weblogic.jws.jms.TopicConnectionFactory
```

```
        --initialNamingFactory=
weblogic.jndi.WLInitialContextFactory

        --url=ccl://localhost:6789/Stream/Default/TestJMS/
instream1
        --host=localhost
        --port=7001
```

Explanation of the adapter properties:

The environment variable **c8.baseHostPort** controls where to establish the connection to the server. This can be set by the -D option on the command line.

The value given for **--topic** (weblogic.jws.jms.MyJMSTopic)is the JNDI Name for the topic destination you want to send the c8 messages to.

The value given for **--factoryName** (weblogic.jws.jms.TopicConnectionFactory) is the JNDI name for the Connection Factory you created, which targets the WebLogic Server where your JMS Server is deployed.

The value given for --initialNamingFactory (weblogic.jndi.WLInitialContextFactory) is the name of the factory class to use to make the connection.

The value given for **--URL** (ccl://localhost:6789/Stream/Default/TestJMS/instream1) is the URL of the Sybase CEP stream you want to subscribe to. The value of the URL can be obtained from Studio, by clicking on the stream in the Explorer View and then clicking on the Properties View.

Make sure that you set the **host** and **port** to be the host and port of the JMS server, not the Sybase CEP Server. The host and port for the Sybase CEP Server are specified as part of the c8.baseHostPort.

g) If the JMSAdapter started successfully, you should see the following output:

```
Connecting to CPX server...
    Opening a subscription...
    Listening for messages...
```

h) Check to make sure that the messages arrive at the topic destination. The method for doing this depends upon the JMS Server you are using.

Shutdown Sequence

Provides directions on shutting down Sybase CEP.

We recommend that you shut down in the following order:

1. Stop the input source.
2. Stop the output adapter.
3. Stop the query.

If the Sybase CEP module is shut down before the output adapter is stopped, the adapter displays error messages. This is a result of the adapter being unable to subscribe to the stream and is not indicative of more serious problems.

Reading, Writing, and Converting Timestamps

This section describes the rules for working with timestamps in Sybase CEP.

Internally, timestamps are always stored as the number of microseconds since midnight January 1, 1970 UTC/GMT. (This time is referred to as "the beginning of the epoch".) This is true for columns of type TIMESTAMP and for the internal row timestamp.

Although internally timestamps are always stored in UTC/GMT, when timestamps are converted to or from a string (for example, "2010-12-31 13:00:00.000000") the time zone is taken into account. The rules used in converting between internal timestamps and strings are shown below:

- If no format is supplied, the value is assumed to be in microseconds since the beginning of the epoch. Such times are ALWAYS assumed to be in UTC/GMT.
- If the format specifier includes a timezone specifier (for example, "YYYY-MM-DD HH24:MI:SS.FF **TZD**") then timestamp values should be supplied with a time zone such "PST" for U.S. Pacific Standard Time, and the time is assumed to be correct for that timezone.
- If the timestamp is supplied as a string, and if a format specifier (for example, "YYYY-MM-DD HH24:MI:SS.FF") is supplied, but does not include a time zone specifier, then the time is assumed to be provided in local time, where local time is determined by the machine on which the program is running.

For example, if you enter the timestamp as 0 microseconds and convert this to U.S. Pacific Standard Time (which is 8 hours behind UTC/GMT) using a format specifier "YYYY-MM-DD HH24:MI:SS TZD" the resulting string will be "1969-12-31 16:00:00 PST".

Generally, you don't need to deal with the internal timestamps, but you should either:

- Use the same time zone (local time) in all your calculations, or
- Specify the time zone for each timestamp that you enter.

If all of your work is done in the same time zone (for example, your company has one office, and all dates and times are entered as local time), you may not need to specify time zone information. However, if you have multiple offices, or if you enter times and dates for customers in different time zones, you need to specify the time zone in the format specifier and in the data that you enter.

These rules apply to all adapters that use, read or write timestamp information as a string.

For more information about time zones, see *Daylight Savings Time and the Time Zone Database* on page 567 in this manual.

For more information about how to specify timestamp formats, see the appropriate appendix in the Sybase CEP *CCL Reference Guide* .

Adapters Supplied by Sybase CEP

This section describes the built-in input adapters and output adapters provided by Sybase CEP.

Below is a table that lists the adapters and the adapter "types". You'll need to know the adapter type if you use an ATTACH ADAPTER statement, for example,

```
ATTACH INPUT ADAPTER Adapter1
 TYPE ReadFromCsvFileAdapterType
 ...
```

In the preceding statement, the type "ReadFromCsvFileAdapterType" is the adapter type for the Read From CSV File adapter.

This table also contains links to the first page describing each adapter.

Note that only in-process adapters can be attached with the ATTACH ADAPTER statement, so only in-process adapters have an adapter type. In the tables below, out-of-process adapters have "N / A" as their adapter type.

Adapter Data Type	Read / Write	Adapter Type (used in ATTACH ADAPTER statement)
Atom	*Atom Reader* on page 389 N / A	AtomReaderAdapterType N / A
Binary file	*Read from binary file* on page 391 *Write to binary file* on page 391	ReadFromBinaryFileAdapterType WriteToBinaryFileAdapterType
Comma-Separated Values (CSV) File	*Read from CSV (Comma-Separated Value) File* on page 391 *Write to CSV (Comma-Separated Value) File* on page 405	ReadFromCsvFileAdapterType WriteToCsvFileAdapterType
Comma-Separated Values (CSV) Socket	*Read from CSV (Comma-Separated Value) Socket* on page 409 *Write to CSV (Comma-Separated Value) Socket* on page 413	ReadFromCsvSocketAdapterType WriteToCsvSocketAdapterType
Database: Poll From Database (see also JDBC)	*Poll From Database* on page 414 *Write to database* on page 423	ReadFromDBAdapterType WriteToDBAdapterType

Adapter Data Type	Read / Write	Adapter Type (used in ATTACH ADAPTER statement)
Database: Read From Database (see also Database, JDBC)	*Read From Database* on page 420 N / A	ReadFromDatabaseAdapterType N / A
Email: Send Email Out (SMTP)	N / A *Send Email Out (SMTP)* on page 464	N / A OpenSourceEmailAdapterType
Email: Java Send Mail	N / A *Java Send Mail* on page 466	N / A N / A
Ganglia	*Ganglia Reader* on page 469 N / A	GangliaReaderAdapterType N / A
Java Messaging Service (JMS)	*JMS Adapter* on page 476 N / A	N / A N / A
JDBC (see also Database)	*JDBC Input* on page 469 *JDBC Output* on page 473	N / A N / A
Random Tuples Generator	*Random Tupless Generator* on page 480 N / A	RandomTuplesGeneratorAdapterType N / A
RAP - The Trading Edition	*RAP Send* on page 482 N / A	PublishToRAPAdapterType N / A
Regular Expression File	*Read from Regular Expression File* on page 485 N / A	ReadFromRegexFileAdapterType N / A
Regular Expression Socket	*Read from Regular Expression Socket* on page 489 N / A	ReadFromRegexSocketAdapterType N / A
RSS	*RSS reader* on page 491 N / A	RssReaderAdapterType N / A
SNMP	*SNMP Set* on page 495 N / A	SNMPGetAdapterType SNMPGetAdapterType
SNMP V1 Trap	N / A *SNMP Send V1 Trap* on page 496	N / A SNMPV1SendTrapAdapterType

Adapter Data Type	Read / Write	Adapter Type (used in ATTACH ADAPTER statement)
SNMP V2 Trap	N / A *SNMP Send V2c Trap* on page 498	N / A SNMPV2SendTrapAdapterType
Sybase RAP	N / A *Sybase RAP* on page 483	N / A N / A
Windows Event Logger	*Windows Event Logger* on page 509 N / A	N / A N / A
XML File	*Read from XML File* on page 500 *Write to XML File* on page 505	ReadFromXmlFileAdapterType WriteToXmlFileAdapterType
XML Socket	*Read from XML Socket* on page 504 *Write to XML Socket* on page 508	ReadFromXmlSocketAdapterType WriteToXmlSocketAdapterType
XML over HTTP	N / A *Write XML Over HTTP* on page 508	N / A WriteXmlOverHttpAdapterType

Adapter	Adapter Type (used in ATTACH ADAPTER statement)
Atom Reader on page 389	AtomReaderAdapterType
Binary: Read from binary file on page 391	ReadFromBinaryFileAdapterType
Comma-Separated Values (CSV): Read from CSV File on page 391	ReadFromCsvFileAdapterType
Comma-Separated Values (CSV): Read from CSV Socket on page 409	ReadFromCsvSocketAdapterType
Database: Poll From Database adapter on page 414	ReadFromDBAdapterType
Database: Read From Database adapter on page 420	ReadFromDatabaseAdapterType
Ganglia Reader on page 469	GangliaReaderAdapterType
JDBC Input on page 469	N / A
JMS Input Adapter on page 476	N / A
Random Tuples Generator on page 480	RandomTuplesGeneratorAdapterType

Adapter	Adapter Type (used in ATTACH ADAPTER statement)
RAP - The Trading Edition on page 482	PublishToRAPAdapterType
Regular Expressions: Read from Regular Expression File on page 485	ReadFromRegexFileAdapterType
Regular Expressions: Read from Regular Expression Socket on page 489	ReadFromRegexSocketAdapterType
RSS reader on page 491	RssReaderAdapterType
SNMP Set on page 495	SNMPGetAdapterType
Windows Event Logger on page 509	N / A
XML: Read from XML file on page 500	ReadFromXmlFileAdapterType

Adapter	Adapter Type (used in ATTACH ADAPTER statement)
Binary: Write to binary file on page 391	WriteToBinaryFileAdapterType
Comma-Separated Values (CSV): Write to CSV File on page 405	WriteToCsvFileAdapterType
Comma-Separated Values (CSV): Write to CSV Socket on page 413	WriteToCsvSocketAdapterType
Database: Write to database on page 423	WriteToDBAdapterType
Email: Java Send Mail on page 466	N / A
Email: Send Email Out (SMTP) on page 464	OpenSourceEmailAdapterType
JDBC Output on page 473	N / A
JMS Output Adapter on page 476	N / A
SNMP Set on page 495	SNMPGetAdapterType
SNMP Send V1 Trap on page 496	SNMPV1SendTrapAdapterType
SNMP Send V2c Trap on page 498	SNMPV2SendTrapAdapterType
XML: Write to XML file on page 505	WriteToXmlFileAdapterType
XML: Write XML Over HTTP on page 508	WriteXmlOverHttpAdapterType

Adapter Properties

Almost all in-process adapters have properties that you can set. In Studio, these properties are displayed in, and can be edited in, the adapter's Properties View after you attach the adapter to a stream. Each of these properties is described in the adapter's .adl file. Inside the .adl file, each

property is referred to as a "parameter". When discussing adapter properties, we sometimes use the words "property" and "parameter" interchangeably, but we will generally favor the term "property" since that is the term shown in Studio and in the ATTACH ADAPTER statement.

For example, in the Read From CSV File adapter, one of the properties that you must set is the name of the file to read from. For each adapter, we include a table that lists the properties for that table. In general, each table contains four columns:

- Property Name (screen): This is the property name that you will see on-screen when you use Studio to edit an adapter's property values.
- Property Name (Attach Adapter): This is the property name that you must use in ATTACH ADAPTER statements. For example, the "Filename" in the ATTACH ADAPTER statement below is `xyz.csv`.

  ATTACH INPUT ADAPTER someName ... PROPERTIES Filename = 'xyz.csv' ...;
- Type: The data type, such as STRING, INTEGER, and so on.
- Description.

In many cases, the property's screen name and Attach Adapter name are the same.

Atom Feed Reader Input Adapter

The ATOM feed reader enables you to get information from ATOM data sources. ATOM data sources enable connections through URL and send their information in a special XML format.

The retrieved information will be inserted into a Sybase CEP stream. The incoming XML information must include:

- feed_title
- feed_link
- feed_author_name
- entry_title
- entry_link
- entry_content

If additional fields exist in the XML file, they are ignored.

The following table describes the properties for the ATOM feed reader:

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
URL	URL	String	The URL of the ATOM data source.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Refresh interval	RefreshInterval	Interval	(Optional). How often to query the specified URL for data. In microseconds, unless qualified with Interval formatting (see "Time Literals" in the Sybase CEP *CCL Reference Guide* for more information). Defaults to 1 minute.
Timestamp Format	TimestampFormat	String	(Optional). The format of the timestamp. See "Timestamp Format Codes" in the Sybase CEP *CCL Reference Guide* for more information. Defaults to "YYYY/MM/DDTHH24:MI:SS".

For an example of how to use the ATOM Feed Reader, see the LiveJournalAlerts example in the examples directory under Sybase CEP *Installation Guide* .

For example, on Microsoft Windows, the directory is typically:

```
C:\Program Files\SybaseC8\Server\examples\Web\LiveJournalAlerts
```

On UNIX-like operating systems, the directory is typically:

```
/home/<username>/sybasec8/server/examples/Web/LiveJournalAlerts
```

If you are not already familiar with the specific XML format used by ATOM, you may find useful information at the following Web site:

http://www.atomenabled.org/developers/syndication/atom-format-spec.php

Binary: Read From Binary File Adapter

A Read From Binary File adapter reads rows from a Sybase CEP binary file. This adapter is used for internal testing only.

Binary: Write To Binary File Adapter.

A Write To Binary File adapter writes rows to a Sybase CEP binary output file. This adapter is used for internal testing only.

We recommend that this adapter not be used.

Comma-Separated Values (CSV): Read From CSV File Adapter.

A Read From CSV File adapter reads Rows from a CSV file.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Filename	Filename	String	The name and path of the CSV file to read data from. Specify a file name and path either by typing it in or by clicking on the Browse button for this field. The CSV file name definition follows the format defined by regular expression. By default, the path is relative to the server's adapters base folder and must be underneath that base folder, but you can use $ProjectFolder to specify a path relative to the project folder. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. For details about the Browse and Edit buttons to the right of the filename, see the discussion following this table.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Loop count	LoopCount	Integer Min: 0 Max: 2000000000 Default: 1	If the Loop Count is greater than 1, the data will be sent the specified number of times. After the adapter finishes reading the file, the adapter will start reading again from the beginning. If the Loop Count is 1, the file is read only once and the adapter stops sending data once the end of file is reached. If the Loop Count is 0, the file is read continuously. If the loop count is left blank, it will default to 1.
Rate	Rate	Float Min: 0.000001	If this property is non-zero, the adapter reads data from the input file at the specified rate per second. Any row timestamps in the input file are not be used to determine the time at which rows are sent. This property takes precedence over the "File has timestamp column" property. The default is to leave this unset, in which case the row timestamp determines the relative timing at which rows are sent.
Set timestamp to current time	UseCurrentTimestamp	Boolean Default: false	If true, the adapter overrides the row timestamp specified in the file with the current system time.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
File has timestamp column	TimestampColumn	Boolean Default: true.	Every row must have a row timestamp that indicates the time the row was created. You can set the row timestamp in the input data (for example, in the CSV file), or this timestamp can be set by the adapter at the time that the row is sent to the server. If "File has timestamp column" is true, the adapter assumes that the first field in each row contains a row timestamp. If this flag is set to false, the adapter inserts the current time into the row timestamp field before sending the row. Note: The column with the row timestamp is not explicitly listed in the schema. See the discussion of "row timestamp" in the Sybase CEP *Programmer's Guide* for more information about row timestamps.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Timestamp column format	TimestampColumnFormat	String	If "File has timestamp column" is set to true, the Timestamp column format specifies the format of that row timestamp column, e.g. 'YYYY/MM/DD HH24:MI:SS.FF TZD'. If no timestamp format is specified, the adapter assumes that the timestamp is represented as a number of microseconds since 00:00:00 Jan 1, 1970 UTC/GMT. If you specify a format specifier, this format specifier applies only to the row timestamp column; it does not apply to all input columns of type TIMESTAMP. For more information, see *Reading, Writing, and Converting Timestamps* on page 384.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Timestamp Base	TimestampBase	Timestamp	If this property is set, when the adapter starts, it behaves as though it has started at the time specified by the Timestamp base. This affects the time that the first row in the input file is sent. For example, if the Timestamp base is 2006-02-01 09:00:00 and the row timestamp of the first row in the input file is 2006-02-01 09:00:05 (in other words, 5 seconds after the timestamp base), the adapter will not send the first row until 5 seconds after the adapter was started regardless of what time the adapter was actually started. This can be used to help synchronize data in multiple streams. If this property is left blank, the first row will be sent immediately after the adapter module starts.
File Has A Title Row	TitleRow	Boolean Default: false	If this flag is true, the adapter assumes that the first line in the file contains column names, in which case the adapter tries to match the column names in the file with the column names in the stream's schema. If this flag is set to false, the adapter assumes that the first line in the file contains a row of data, in which case the order of the CSV columns must be the same as the order of the Row descriptor fields.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Field Separator Character	CsvDelimiterString	String Default: , (comma)	This indicates which character(s) separate the fields in the file. In most cases, this should be left at the default value, which is a comma. However, you can specify different separator character(s), such as a vertical pipe symbol ("|") and so on. If you specify multiple characters, each of those characters is treated as a separator; the adapter does not look for a multi-character separator.
Line Continuation Character	CsvLineContinuationCharacter	String Default: ^ (caret)	If a single row of data must be split across multiple input lines, you can specify a line continuation character that tells the adapter to treat the multiple lines as a single row. By default, this line continuation character is the caret ("^") symbol. Note that this character is treated as the line continuation character only if it is the last non-white-space character preceding a newline. See the discussion below for an example. If you specify multiple characters, each of those characters is treated as a line continuation character; the adapter does not look for a multi-character line-continuation indicator. If the line continuation field is empty, no line continuation will be possible in the CSV file. This allows data with wildly varying input characters to be properly processed.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
String Quote Characters	CsvQuoteCharacters	String Default: ' (single quote) and " (double quote)	By default, both the single quote and the double quote characters may be used to delimit a string. If you specify multiple characters, each of those characters is treated as a quote character; the adapter does not look for a multi-character quote indicator. If the string is started with a double quote character, it must end with a double quote character, and single quote characters within that string are treated as regular characters rather than as the end of the string. Similarly, if the string is started with a single quote character, it must end with a single quote character, and any double quote characters within the string are treated as normal characters. If you change the defaults, only the character(s) that you specify are treated as quote characters.
NULL string value	CsvNullString	String Default: NULL	This allows you to specify what value indicates a NULL value in the file. The value should not be quoted. If the NULL string value is the word NULL (without quotes), the value "NULL" (with quotes) is a 4-letter string, not an indication that you want to use a NULL value. If you do not specify a NULL string value, then an "empty" string field where there is nothing between the field separator characters for that field is treated as NULL.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Line Terminator Character	CsvLineTerminatorChar	String Default: \n	This allows you to specify what character represents the end-of-line character. You specify the character in one of the following ways: `\n` This 2-character sequence represents the Line Feed character (ASCII 10). `\r` This 2-character sequence represents the Carriage Return character (ASCII 13). `\t` `<any printable char>` This 1 character may be any printable character except the backslash. Enter the character without any quotation marks and without any leading backslash. `\x##` , where "#" represents a hexadecimal digit (0-9, A-F). To indicate that you want to use ctrl-Z as the end-of-line character, you would specify the 4-character sequence `\x26`.
Escape Characters	CsvEscapeCharacters	String Default: \	The characters used to escape the special meaning of other characters, such as column separators and quotes.
Enable DL (Dynamic Loading)	EnableDynamicLoading	Boolean Default: False	When checked, the CSV input adapter uses a dynamic loading function. When unchecked, the CSV input adapter uses the original behavior to read data from a specified file. When enabled, the Filename is ignored.
DL Source Directory	Source Directory	String	The full path of source directory where the CSV file(s) get dynamically loaded into the CSV adapter.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
DL File Name Format	DLFileNameFormat	String	The regular expression definition for the CSV file names to be loaded up.
DL Check Interval	CheckInterval	Long Min: 1 Default: 60	The next check interval for the adapter when no more files need to be processed, in seconds.

Note: The CSV files will be loaded up periodically from an accessible folder that has been predefined, in the order of the creation date time stamp.

Each input stream has a property that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter (see the stream's Properties tab in). If this stream property is set to true, it overrides any row timestamp set by the adapter.

The Adapter Properties screen in Sybase CEP Studio allows you to specify an input path and file name for the file by clicking the Browse button and identifying the file. The path is relative to the adapters base folder, either as specified for Sybase CEP Server running on the same computer as Sybase CEP Studio, or as specified in the preferences for Sybase CEP Studio. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. In order for this feature to work properly, make sure that the Adapter's Base Folder field in Sybase CEP Studio Settings is set to the same folder as the adapters base folder for Sybase CEP Server. The Base Folder setting for the Sybase CEP Server is specified during the installation process, and can be changed in Sybase CEP Server's `c8-server.conf` file. The base folder setting for Sybase CEP Studio can be set from the Tools->Settings command on the Sybase CEP Studio menu.

The Edit button opens an editor that will allow you to edit any file whose name you enter into the filename field. This allows you to correct errors in the data. If the file's extension is "csv" or "xml", Sybase CEP Studio opens the appropriate editor specified in the "External Tools" tab available from the menu item "Tools -> Settings". For files with other extensions, on Microsoft Windows the editor is the one specified by the operating system's file associations. On UNIX-like operating systems, Sybase CEP Studio opens the editor specified by the EDITOR environment variable.

Note: When you click the Edit button, Sybase CEP Studio will look for the file in the SybaseC8Repository, even if the adapters base folder is set to another location. you can need to use the Browse button (adjacent to the Edit button) to navigate to the desired directory before you try to edit the file.

NULL Values

Describes how to work with NULL values.

To specify a NULL value, simply do not put any value into that field of the row. For example, in the 3 rows below, the first row will have a NULL value for the first column; the second row will have a NULL value in the second column, and the third row will have a NULL value in the third column.

,2,3

1,,3

1,2,

Title Rows and the Hidden Row Timestamp Column

The ReadFromCSVFile adapter enables input data to contain an optional title row, which lists the names of the columns of data.

To ensure that the number of column names in the title row matches the number of columns in the data, you must enter a placeholder name in the title row if your input data contains a hidden row timestamp column. For example, if your data contains a name column, an ID column, and a hidden row timestamp column, then the title row might look similar to the following:

```
Ts,Name,ID
```

The name "Ts" is a placeholder. The name must follow the usual rules for column names, but the name is not used in any way. The name does not appear in the stream schema, and you cannot use the name in queries. Since the hidden row timestamp column must be the first column, the placeholder name must be the first name in the title row.

Other Time-Related Properties

Rate

The Rate property allows you to specify the rate (in rows per second) at which the server reads and processes rows. If you specify it, the minimum value for Rate is 0. The maximum rate you can enter is approximately 2 billion, but the practical maximum depends upon the speed of your computer.

Furthermore, the exact time that rows are sent depends on low-level factors such as task switching, so the actual rate at which rows are sent only approximately the same as the rate that you specify.

If Rate is set, the server will act as though "Set timestamp to current time" is also set. In other words, the actual time that the row is sent overwrites the row timestamp in the input file. Setting the rate does not change the row timestamps in the rows.

File has timestamp column

If you set the "File has timestamp column" property to true, the first column will be treated as the row timestamp column.

When the Read From CSV File adapter starts running, it sends the first row in the file immediately. Subsequent rows are sent based on the difference between row timestamps. For example, if the second row's timestamp is 5 seconds after the first row's timestamp, then the second row will be sent 5 seconds after the first row unless another property, such as Rate, overrides this. If you are using the Accelerated Playback feature, the actual time that the second row is sent depends on the Accelerated Playback rate.

Note: The Rate property and Timestamp base property override or alter the behavior of the File has timestamp column property. See the descriptions of those properties for details.

If you set Has title row to true, the first row of the input file must include a name for the row timestamp column, even though that column name is not used in any way. For example, if your stream has three columns, row timestamp, department number, and department name, and if File has title row is set to true, then the first line of your input file should look similar to:

```
dummyTsColName,DeptNum,DeptName
```

where dummyTsColName may be any string that is valid as a column name.

Timestamp format column

The "Timestamp format column" property allows you to specify the format of the row timestamp column (for example, 'YYYY-MM-DD HH24:MI:SS.FF'). Since the row timestamp column is of type TIMESTAMP, the format may be any of the formats that are valid for columns of type TIMESTAMP. (See the CCL Reference for a description of the valid formats for TIMESTAMP data type. See *Daylight Savings Time and the Time Zone Database* on page 567 and *Reading, Writing, and Converting Timestamps* on page 384 for more information about time zones.)

The Timestamp format column property should only be set to a value if you set the property File Has Timestamp Column to true.

If you do not set the Timestamp Format Column, the server assumes that the row timestamp is specified in microseconds since midnight January 1, 1970 GMT.

Timestamp Base

The timestamp base enables you to minimize a common problem; if you have multiple input files that are related, the first row in each file may not have the same starting time as the first row in all the other files.

The adapters do not communicate directly with each other, and since the adapters do not control the exact time at which they start running or the order in which the operating system gives them time slices, so if each adapter sends its first row as soon as it can, rows may arrive out of order even though the rows all have correct row timestamps.

Imagine a simple case in which the first row of file F1 has a row timestamp of 2006-02-01 09:00:00 and the first row of file F2 has a row timestamp 5 seconds later, in other words, 2006-02-01 09:00:05. When you start processing the data, you want to tell F2's input adapter to stall 5 seconds relative to F1's input adapter. To tell F2's input adapter to stall, specify the same timestamp base for both adapters. For example, if you tell the adapter reading file F1 to use a timestamp base of 2006-02-01 09:00:00, then that adapter will send its first row (with timestamp 2006-02-01 09:00:00) as soon as it can. If you tell the adapter reading file F2 to use the same timestamp base (2006-02-01 09:00:00), then that adapter will wait approximately 5 seconds before sending its first row, which has a timestamp of 2006-02-01 09:00:05.

Note: If you are using timestamp base with the Accelerated Playback feature, the actual delays will be adjusted to factor in the accelerated playback rate.

The value in the timestamp base field is interpreted according to the setting in the Timestamp column format (for example, "YYYY-MM-DD HH24:MI:SS.FF") if that field is set.

Note: Timestamp base is ignored if "Rate" is set.

Set Timestamp to Current Time

If Set timestamp to current time is set to true, Sybase CEP sets each row's timestamp to the current system clock date and time, even if the row already has a row timestamp. If Set timestamp to current time is set to no, Sybase CEP leaves the row timestamp unchanged.

If the data does not already have a row timestamp, you should set Set timestamp to current time to true.

The Set timestamp to current time property does not control the rate at which rows are sent. You should specify either the Rate or set File has timestamp column to true to control when rows are sent.

It is possible to use existing row timestamp values in the file to control when rows are sent, but then overwrite the row timestamp with the current time. This is useful if you want to send data in the file at the same rate as the data was originally created, but you want the actual row timestamps to be set to the current time. For example, suppose that the original row timestamps of the first 3 rows were:

2006-02-01 09:00:00.000000

2006-02-01 09:00:05.000000

2006-02-01 09:00:10.000000

In other words, the rows timestamps are 5 seconds apart. If you want to process the rows using the current date and time, but still have the rows appear to arrive 5 seconds apart, then specify both File has timestamp column and Set timestamp to current time. Also, do not set the Rate property, since Rate overrides File has timestamp column.

If you also set Timestamp base in addition to setting Set timestamp to current time to true and File has timestamp column to true, then the adapter sends the first row based on the difference

between its row timestamp and the timestamp base, and subsequent rows will be sent based on the difference in row timestamps. If the row timestamps are 5 seconds apart then the rows will be sent 5 seconds apart.

Loop Count

This section describes typical behaviour for loop counts in Sybase CEP.

The ReadFromCSVFile adapter always has a 3-second startup delay before the first row is sent.

For CSV files, if a user has Timebase starts at, then a further delay may be inserted, depending upon user supplied data and property settings.

If looping is used, the 3-second delay does not happen after the initial delay, but the Timebase starts at does happen on each loop.

Note: If you choose to loop through the data repeatedly, and if the data contains row timestamps, the row timestamps will be adjusted on the 2nd and subsequent iterations. Otherwise, you would get an error message about rows being out of order when the first row was read in again after the last row had been processed.

Line Continuation Character

Each row is on a single line, which is terminated by a newline. This section describes how to split information over multiple lines with the line continuation character.

The ReadFromCSVFile adapter provided by Sybase CEP treats both a bare LineFeed (ASCII 10) and the sequence CarriageReturn+LineFeed (ASCII 13 followed by ASCII 10) as newlines. In some cases, you want to split lengthy data onto multiple lines. To indicate that the next line is part of the current line, use a line continuation character.

Note that the character is interpreted as a line continuation character ONLY when it is the last non-whitespace character prior to a newline. For example, consider the following data:

Row1, "The caret character (^) is treated as a line continuation ^

character only when it is at the end of a line."

Row2, "This row is the second row."

The first caret on the first line is treated as part of the data. Only the last caret is treated as a line continuation character.

Cautions on Using Quote Characters, Line Continuation Characters, and Field Separator Characters

Rules to follow when using ASCII characters.

You may use almost any ASCII character(s) that you want for each of these purposes; however, follow the rules listed below:

- The set of characters for one purpose should not overlap the set of characters for any other purpose. For example, you should not use the same character as both a field separator and a line continuation character.
- Do not use the newline character (or individual Line Feed or Carriage Return characters) as quote character(s), line continuation character(s), or field separator character(s).
- As mentioned above, when you specify more than one character, EACH of the characters that you specify will be used for the specified purpose. You cannot create multi-character strings for any of these purposes; for example, if you specify "][" as the field separator characters, then each of the individual characters will be treated as field separators. In other words, the sequence "][" will be treated as 2 separate field separators.
- There is currently no "escape character" that indicates that the next character should be treated as a literal rather than as a special character.

Format Of INTERVAL Values In The CSV File

Use any acceptable format for INTERVAL values inside a CSV or XML file.

The following are acceptable formats for INTERVAL values:

- a 64-bit signed integer representing a number of microseconds.
- [D [day[s]]][ ][HH:MI[:SS[.FF]]] (for example, 1 02:03:04.000005).
- [D day[s]][ ][HH hour[s]][ ][MI minute[s]][ ][SS[.FF] second[s]] (for example, 1 day 2 hours 3 minutes 4.000005 seconds).

For a complete list of valid formats for INTERVAL values, see 'INTERVAL Literals" in the Sybase CEP *CCL Reference Guide* .

Note: Although CCL statements require the keyword INTERVAL prior to the value for some of these forms, no INTERVAL keyword is required when using such values in CSV or XML files.

Comma-Separated Values: Write To CSV File Adapter.

A Write To CSV File adapter writes received rows to a CSV output file.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Filename	Filename	String	The name and path of the CSV file to write data to. You can specify a file name and path either by typing it in or by clicking on the Browse button for this field. The CSV file name definition follows the format defined by regular expression. By default, the path is relative to the server's Base Folder and must be underneath that base folder, but you can use $ProjectFolder to specify a path relative to the current project folder. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. For more information about the Browse and Edit buttons, see the discussion that follows this table.
Maximum Size in bytes	MaximumSize	Integer Min: 1	The maximum size of the output file. If this property is set, the adapter starts writing a new file every time the size of the current output file becomes greater than this property. The files are named <filename>, <filename>.001, <filename>.002, and so on, where <filename> is the value of the Filename property.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Append To Existing File	Append	Boolean	This parameter affects how the adapter behaves when the adapter re-starts. In all cases, when the adapter writes to a file and the size of the file reaches the size specified in the MaximumSize parameter described above, the adapter renames <Filename> to <Filename>.###, where "###" is the next available number. The adapter then opens another file named <filename> and writes to it. If "Append To Existing File" is set to False, then when the adapter is (re-)started, the adapter moves any existing file named <Filename> to <Filename>.### and starts a new file named <Filename>. If a maximum size is specified and "Append To Existing File" is set to True, then when the adapter is (re-)started, the adapter appends to any existing file named <Filename> rather than immediately renaming the existing file.
Set timestamp to current time	UseCurrentTimestamp	Boolean	If set to true, the adapter overrides the timestamp specified in the row with the current system time. Defaults to false.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Write title row	TitleRow	Boolean	If set to true, the adapter adds a title row to the output. The title row lists the column names.
Write timestamp column	TimestampColumn	Boolean	If set to true, the adapter writes the message timestamps in the first column of the file.
Timestamp column format	TimestampColumnFormat	String	The Timestamp column format specifies the format of the row timestamp column, for example, 'YYYY/MM/DD HH24:MI:SS.FF TZD'. If no timestamp format is specified, the adapter assumes that the timestamp is represented as a number of microseconds from 00:00:00 Jan 1, 1970 UTC/GMT. For more information, see *Reading, Writing, and Converting Timestamps* on page 384.
Flush each row	FlushAfterEachRow	Boolean	If set to true, the adapter writes each row to disk as soon as possible. If set to false, the adapter can wait until multiple rows are ready to be written to disk and then write them as a group. Flushing after each row may reduce overall performance.
CSV delimiter string	CsvDelimeterString	String	The character that separates columns in the output. Defaults to a comma (,).

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Enable DL (Dynamic Loading)	EnableDynamicLoading	Boolean Default: False	When enabled, the CSV input adapter will use a dynamic loading function and the filename will be ignored. When unchecked, the CSV input adapter will use the original behavior to read data from a specified file.
DL Source Directory	Source Directory	String	The full path of source directory where the CSV file(s) get dynamically loaded into the CSV adapter.
DL File Name Format	DLFileNameFormat	String	The regular expression definition for the CSV file names to be loaded up.
DL Check Interval	CheckInterval	Long Min: 1 Default: 60	The next check interval for the adapter when no more files need to be processed, in seconds.

Note: The CSV files are loaded up periodically from an accessible folder that has been predefined, in the order of the creation date time stamp.

The Adapter Properties screen in Sybase CEP Studio allows you to specify an input path and file name for the file by clicking the Browse button and identifying the file. The path is relative to the adapters base folder, either as specified for Sybase CEP Server running on the same computer as Sybase CEP Studio, or as specified in the preferences for Sybase CEP Studio. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. In order for this feature to work properly, make sure that the Adapter's Base Folder field in Sybase CEP Studio Settings is set to the same folder as the adapters base folder for Sybase CEP Server. The Base Folder setting for the Sybase CEP Server is specified during the installation process, and can be changed later in the Sybase CEP Server's `c8-server.conf` file. The base folder setting for Sybase CEP Studio may be set from the Tools->Settings command on the Sybase CEP Studio menu.

The edit button brings up an editor that will allow you to view any file whose name you enter into the filename field. If the file's extension is "csv" or "xml", Studio brings up the appropriate

editor specified in the "External Tools" tab available from the menu item "Tools -> Settings". For files with other extensions, on Microsoft Windows the editor is the one specified by the operating system's file associations and on UNIX-like operating systems Sybase CEP Studio opens the editor specified by the EDITOR environment variable.

Note: When you click the Edit button, Sybase CEP Studio looks for the file in the SybaseC8Repository, even if the adapters base folder is set to another location. You may need to use the Browse button (adjacent to the Edit button) to navigate to the desired directory before you try to open the file.

Unless specified otherwise, the format of TIMESTAMP values (including the row timestamp) inside the CSV file is a 64-bit signed integer representing the number of microseconds since the beginning of the epoch (midnight January 1, 1970 UTC/GMT).

When Sybase CEP Server writes INTERVAL values in CSV format, the values are always written as a 64-bit signed integer representing a number of microseconds.

Comma-Separated Values (CSV): Read From CSV Socket Adapter.

A Read from CSV Socket adapter opens a TCP connection to a given address, specified through Host and Port properties. Once the connection is established, the adapter reads rows from this connection as comma-separated values.

If a connection is lost during adapter execution, it attempts to reconnect.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Host	Host	String	Host name or IP address of the data source.
Port	Port	Integer Min: 1 Max: 65535 Default: none	Port number of the data source.
Data Has Title Row	TitleRow	Boolean Default: True	Whether to expect a title row.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Data has timestamp column	TimestampColumn	Boolean Default: True	Every row must have a row timestamp that indicates the time that the row was created. This row timestamp may be set by the system at the time that the row arrives, or it can be specified by the user (for example, if running a simulation). If the Data has timestamp column flag is true, the adapter assumes that the first column in each row contains a row timestamp and that this row timestamp should be used to control the timing of when the rows are sent to the server. If this flag is set to false, the adapter inserts the row's arrival time into the row timestamp column, and sends the data as quickly as it can. If a row timestamp is supplied, the adapter assumes that the timestamp is represented as a number of microseconds from 00:00:00 Jan 1, 1970 UTC/GMT.
Timestamp Column Format	TimestampColumnFormat	String	The format in which timestamp values are stored, for example, YYYY/MM/DD HH24:MI:SS.FF. If blank, timestamp values are in microseconds since January 1, 1970, 12:00:00 AM.
Field Separator Character	CsvDelimiterString	String Default: , (comma)	This indicates which character(s) separate the fields in the data. For most cases, leave this at the default value, which is a comma. However, you can specify a different separator character, such as a vertical pipe symbol ("|") etc. Note that if you specify multiple characters, each of those characters is treated as a separator; the adapter does not look for a multi-character separator.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Line Continuation Character	CsvLineContinuationCharacter	String Default: ^ (caret)	If a single row of data must be split across multiple input lines, you can specify a line continuation character that tells the adapter to treat the multiple lines as a single row. By default, this line continuation character is the caret ("^") symbol. This character is treated as the line continuation character only if it is the last non-white-space character preceding a newline. If you specify multiple characters, each of those characters is treated as a line continuation character; the adapter does not look for a multi-character line-continuation indicator. If the line continuation field is empty, no line continuation is possible. This allows data with wildly varying input characters to be properly processed.
String Quote Characters	CsvQuoteCharacters	String Default: ' (single quote) and " (double quote)	By default, both the single quote and the double quote characters may be used to delimit a string. If you specify multiple characters, each of those characters is treated as a quote character; the adapter does not look for a multi-character quote indicator. If the string is started with a double quote character, it must end with a double quote character, and single quote characters within that string are treated as regular characters rather than as the end of the string. Similarly, if the string is started with a single quote character, then it must end with a single quote character, and any double quote characters within the string are treated as normal characters. If you change the defaults, then only the character(s) you specify are treated as quote characters.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
NULL string value	CsvNullString	String Default: NULL	This allows you to specify what value indicates a NULL value in the file. The value should not be quoted. If the NULL string value is the word NULL (without quotes), then the value "NULL" (with quotes) is a 4-letter string, not an indication that you want to use a NULL value. If you do not specify a NULL string value, then an empty string field, where there is nothing between the field separator characters for that field, is treated as NULL.
Line Terminator Character	CsvLineTerminatorChar	String Default: \n	This allows you to specify what character represents the end-of-line character. You specify the character in one of the following ways: `\n` This 2-character sequence represents the Line Feed character (ASCII 10). `\r` This 2-character sequence represents the Carriage Return character (ASCII 13). `\t` This 2-character sequence represents the Tab character (ASCII 9). `<any printable char>` This 1 character may be any printable character, for example, '$', '|', etc., except the backslash. Note that you enter the character without any quotation marks and without any leading backslash. `\x##` , where "#" represents a hexadecimal digit (0-9, A-F). Thus, for example, to indicate that you want to use ctrl-Z as the end-of-line character, you would specify the 4-character sequence `\x26`

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Escape Characters	CsvEscapeCharacters	String Default: \	The characters used to escape the special meaning of other characters, such as column separators and quotes.

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter

Comma-Separated Values (CSV): Write To CSV Socket Adapter.

A Write to CSV Socket adapter opens a TCP connection to a given address, specified through Host and Port properties. Once the connection is established, the adapter writes Rows into this connection as comma-separated values.

If a connection is lost during adapter execution, it attempts to reconnect.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Host	Host	String	Host name or IP address of the data destination.
Port	Port	Integer	Port number of the data destination.
Write title row	TitleRow	Boolean	Whether to send title row (default: "true").
Write timestamp column	TimestampColumn	Boolean	If set to true, the adapter writes the message timestamps in the first column of the file. The timestamp is represented as a number of microseconds from 00:00:00 Jan 1, 1970 UTC/GMT.
Timestamp Column Format	TimestampColumnFormat	String	The format in which timestamp values are stored, e.g., YYYY/MM/DD HH24:MI:SS.FF If blank, timestamp values are in microseconds since January 1, 1970, 12:00:00 AM.

CSV delimiter string	CsvDelimeterString	String	The character that separates columns in the output. Defaults to a comma (,).

Database: Poll From DB Input Adapter

The Poll From Database adapter reads information from a table on an external database server.

The adapter reads the table repeatedly, at intervals specified by the polling interval property. You can choose to read the entire table each time, or read only the rows that have been added since the last time that you read the table.

This adapter is an in-process adapter. The server starts it automatically when necessary. You attach this adapter to a stream by using commands inside Sybase CEP Studio, and properties for this adapter are set using Sybase CEP Studio. Alternatively, you may use the ATTACH ADAPTER statement to attach the adapter to a stream and specify the values of the adapter properties.

The adapter properties are listed in the table below:

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Query	Query	String	Required. This is the SQL query (for example, SELECT statement) that you want to run on the external database server. The order of fields that you retrieve should match the order of the fields in the stream to which you are writing. For example, the first field in the SELECT list of the SQL query should correspond to the first field in the schema of the stream.
DBName	DBName	String	Required. This indicates which database to execute the query on. This is the name of a service that is defined in the `c8-services.xml` file, not the name of the database. The service information will include information about how to connect to the external database server and the name of the database to connect to.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Poll Interval	PollInterval	Interval	Required. This indicates how frequently Sybase CEP Server should check the external database server for changes. The value can be a number of microseconds, or you can use standard INTERVAL values such as 100 milliseconds or 1 hour.
Counter field	CounterField	Integer or Long	This is the name of the counter/sequence number field. In the Sybase CEP schema, this is not the name of the field in the external table. The column's data type should be compatible with Sybase CEP's INTEGER or LONG data type.
Counter field initial value	CounterFieldInitValue	Integer or Long	If this field is set, then the first time that the query is executed, the adapter retrieves only records whose counter field contain values greater than the value specified in this Initial Value field. Data values equal to the Initial Value are not retrieved.) The initial value of the Counter Field may be either a LONG or an INTEGER, depending on the data type of the Counter Field.
Timestamp Field	TimestampField	String	This is the name of the timestamp/datetime column. In the Sybase CEP schema, this is not the name of the field in the external table. The column's data type should be compatible with Sybase CEP's TIMESTAMP data type. See *Datatype Mappings* on page 347 for tables that show which database data types correspond to Sybase CEP's TIMESTAMP data type. The actual data type may have a resolution less than or equal to TIMESTAMP. If the resolution is greater than TIMESTAMP (for example, nanosecond vs. microsecond), you may get occasional cases of missing or duplicate records.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Timestamp field initial value	TimestampFieldInitValue	Timestamp	If this field is set, then the first time the query is executed, the adapter retrieves only records whose timestamp field contains values greater than the value specified in this Initial Value field. Data values equal to the Initial Value are not retrieved.

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter

The query is usually a SELECT statement that will read values from the external database server.

For example, the query might look like:

```
SELECT col2 FROM StockTable WHERE col1 = 'IBM';
```

The query can also contain a statement other than SELECT. For example, you can call a stored procedure that returns values.

The Database Name property indicates which database to connect to. Despite the fact that we call this property the "Database Name", it is actually the name of a service which must be configured in the `c8-services.xml` file. The service name is not necessarily identical to the name of the database. Each service has information about how to connect to the external database server and the name of the database on that server. For more information about configuring a service to provide access to a database on an external database server, please see the Sybase CEP *Installation Guide* .

The poll interval specifies how frequently to read data from the external database server. You can set property values to specify whether you want the entire table re-read each time, or whether you want to read only the records added since the last time the table was read.

To read the entire table each time, leave the Counter Field, Counter Field Initial Value, Timestamp Field, and Timestamp Field Initial Value empty. This causes the adapter to read the entire external table at an interval specified by the Poll Interval property.

To read just the records that were added since the last time the table was read, please read the section below.

Retrieving a Subset of Records

This section describes how to access only a certain section of records in a table using the Poll from Database adapter.

Normally, the query specified in the Query property will retrieve all the records in the table, or all records that meet your specific criteria if you use a WHERE clause. If the table is large, then retrieving all the rows may consume a lot of resources.

To save resources, the Poll From DB adapter allows you to specify that you want to read only the records that were added since the last time that the query was run. To do this, you need to tell the adapter which column to look at to determine which records are new. This column must be either a date/time column (for example, the date and time at which a stock trade was executed) or some type of counter or sequence number that is assigned in ascending order by the data source.

For example, suppose you are querying a table that contains stock information, in which each row has a column named TradeTime that indicates the time at which the stock trade took place. Suppose also that the previous execution of the query was at 10:00 July 1, 2006, and at 10:05 you want to retrieve just the stock trades that were executed since 10:00. Conceptually, you could do this by modifying your WHERE clause to look similar to:

```
...WHERE... TradeTime > TO_TIMESTAMP("2006-07-01 10:00:00")...
```

Each time that the query is run, the value "2006-07-01 10:00:00" must be replaced with the time of the query's most recent previous run. Since the adapter reads the rows from the external table, the adapter itself can keep track of the highest value read so far and plug that in each time that the query is executed. You must specify which column contains the timestamp or counter.

To specify which column to use, enter a column name into either of two adapter properties, named Timestamp Field and Counter Field. You must also modify your query to refer to a variable stored by the adapter itself; this variable holds the highest timestamp or the highest counter read so far. You must also include an ORDER BY clause that refers to the column name you entered into the Timestamp Field or Counter Field property.

The name of the column you enter must be the name of the column in the CCL schema, in the external table. For example, if you decide to use the TradeTime column of the external table, and if the TradeTime column of the external table is read into the TimeOfTrade column in the CCL stream, then you would specify TimeOfTrade in the Timestamp Field property of this Poll From DB adapter.

Using our example of a table of stock trades, the SQL Query would look like:

```
SELECT ...
FROM StockTable
WHERE TradeTime > ?C8_TIMESTAMP_FIELD ...
ORDER BY TradeTime ...;
```

The Timestamp Field (or the Counter Field, if you use a Counter Field) must be a column that is always higher for newer records. For example, if you were reading from a table of employees, you would want to use a timestamp that indicated each employee's hiring date and time, not birthday. If you used birthdays, then the timestamps in new rows would not necessarily be greater than the timestamps in the most recently read row, and thus a query that retrieved only rows with newer timestamps would not necessarily retrieve all of the new employees.

If the external table does not have an appropriate timestamp/datetime column, you may be able to use a column that acts like a counter or a sequence number that has values that are unique and that ascend over time. In some cases, the table's primary key column may meet these requirements.

For example, suppose that you ship packages to customers, and each package is given a unique serial number. Serial numbers increase over time, and rows are inserted into the table in order by serial number. The serial number is in a column named "SerialNum" and the column name is the same in the external table and the Sybase CEP stream. In that case, set the adapter's property named Counter Field to "SerialNum", and write your query to look similar to:

```
SELECT ...
FROM Shipments
WHERE SerialNum > ?C8_COUNTER_FIELD ...
ORDER BY SerialNum ...;
```

The variable `?C8_COUNTER_FIELD` will be replaced with the most recent value of the Counter Field column that was read during the previous Poll From DB operation. For example, if the previous operation read rows with serial numbers 200-220, then the WHERE clause above would be sent to the external database server as

```
WHERE SerialNum > 220 ... ;
```

The server uses the value of the Timestamp Field or the Counter Field in the most recently read row, not the largest value seen so far. Therefore, you must include an ORDER BY clause to ensure that the values are returned in order and thus that the most recently read row contains the largest value.

If you are reading rows incrementally rather than reading all rows every time the query executes, you must decide whether the very first read should get all rows in the table or only the rows since a particular cutoff point. The adapter properties named "Timestamp Field Initial Value" and "Counter Field Initial Value" allow you to specify this.

If you want to read all of the rows in the table, set the ...Initial Value property to a value lower than the lowest value in the table.

If you want the first execution of the query to return only the rows that were added to the table after a particular cutoff point, then enter that cutoff point into the ...Initial Value property.

Note that the ... Initial Value must be set to a value LOWER than, not equal to, the first value you want to retrieve. For example, if you want to retrieve all rows with a timestamp at or

greater than 9:00 AM July 1, 2006, then the initial value should be "2006-07-01 08:59:59.999999", not "2006-07-01 09:00:00.000000".

The first time you query the table, you probably want to get all of the records in the table, in which case you should set the "...InitialValue" to a value lower than the lowest value in the table.

The following is a complete example:

The table named Shipments has a column named TimeOfShipment which has a data type compatible with 's TIMESTAMP data type. The corresponding column in the stream is named ShippingTime and is of type TIMESTAMP. Each time that a shipment is made, a new record is added to the table, and the TimeOfShipment field of that record is set to the actual time that the item was shipped. The table has data going back as far as January 1, 1998, but you only want to use data data from the year 2000 onward. To access this table with the Poll From DB adapter, you set the following:

- Set the adapter property Timestamp Field to the stream column name ShippingTime.
- Set the adapter property Timestamp Field Initial Value to "1999-12-31 23:59:59.999999").
- Set the query's WHERE clause to include:

```
"... TimeOfShipment > ?C8_TIMESTAMP_FIELD..."
```

Incremental retrievals work well when the only changes to a table are INSERTs. If there are any UPDATE and DELETE operations, they may require special handling. For example, suppose that you are using incremental retrievals on a table that contains stock transactions, and the Timestamp Field for that table is the TradeTime column, in other words, the time at which the trade was executed. If the record is updated to correct the name of the purchaser, for example, the adapter will not see that change unless the TradeTime is also updated, which would be inappropriate if the TradeTime itself was correct. The adapter therefore would not see the new value and would still cache the old value for that particular stock trade. Similarly, if a row were deleted, there might not be any indication of that sent to Sybase CEP Server. If the database that you connect to implements UPDATE as a DELETE plus an INSERT, then you will see the new/updated values, but you won't know that the old values have been deleted. Thus Sybase CEP's local cache of the external table's contents has both the original value and the new value. For this reason, use incremental retrievals only with tables that have INSERTs, not UDPATEs or DELETEs.

Note: The dates from the external database server are assumed to be in UTC/GMT not local time.

If you use incremental retrievals, then since the values in the Counter Field or Timestamp Field must be assigned in ascending order chronologically, the values must not include any NULL values.

In addition to the C8_ROWCOUNT and C8_TIMESTAMP variables, there is also a C8_INVOCATIONS variable that stores the number of times that the adapter has invoked the

query. This is useful in debugging, or if you want the adapter to run for only a finite amount of time or a finite number of executions.

Database: Read From DB Input Adapter

The Read From Database adapter reads information from a table on an external database server and sends that data to a stream.

This adapter has some of the characteristics of the Poll From Database adapter and some of the characteristics of the Read From CSV File adapter. The table below compares and contrasts some key characteristics of these three adapters.

Poll From Database adapter	Read From Database adapter	Read From CSV File adapter
This adapter is used primarily for processing live data.	This adapter is used primarily for replaying historical data or running simulations, rather than for processing live data.	This adapter is used primarily for replaying historical data or running simulations, rather than processing live data.
This adapter does not support Accelerated Playback.	This adapter supports Accelerated Playback.	This adapter supports Accelerated Playback.
You specify the entire database query.	You specify the database table name (or view name) and the WHERE clause.	You specify the name of a file that contains the data.
You can specify the order of the records by using an ORDER BY clause.	The adapter automatically includes an ORDER BY clause that orders by the Timestamp Field. See below for a description of the Timestamp Field property for this adapter.	The order is determined by the physical order of the rows in the data file and by the row timestamps in the data file.

See "Sybase CEP Engine Third-Party Software Dependencies" in the *Sybase CEP Installation Guide* for information about supported databases.

This adapter is an in-process adapter. The server starts it automatically when necessary. Attach this adapter to a stream by using commands inside Sybase CEP Studio. Properties for this adapter are set using Sybase CEP Studio. Alternatively, use the ATTACH ADAPTER statement to attach the adapter to a stream and specify the values of the adapter properties.

The adapter properties are listed in the table below.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description	Optional/ Required
DBName	DBName	String	Indicates which database to execute the query on. This is the name of a service that is defined in the `c8-services.xml` file, not the name of the database. The service information includes information about how to connect to the external database server and the name of the database to connect to.	Required
Table or View	Table	String	The table from which to retrieve the data. This can be a table or a view.	Required
Where Clause	WhereClause	String	This clause limits the result set. This WHERE clause is applied on the remote database server, and can reference any of the fields in the remote table, even if those fields are not also in the schema of the stream into which the data is inserted. You should omit the keyword WHERE. For example, enter "X = Y" rather than "WHERE X = Y".	Optional
Loop Count	LoopCount	Integer	How many times to loop through this data. If the field is left empty, the default value is 1.	Optional
Rate	Rate	Float	How fast to read the records. If this field is not set, the timing of the data is based on the Timestamp field.	Optional

Property Name (screen)	Property Name (Attach Adapter)	Type	Description	Optional/ Required
Timestamp Column	TimestampColumn	String	This is the name of the timestamp/ datetime column. This is the name of the field in the external table, not the name of the field in the Sybase CEP schema. This column's data type should be compatible with Sybase CEP's TIMESTAMP data type. See *Datatype Mappings* on page 347 for tables that show which database data types correspond to Sybase CEP's TIMESTAMP data type. The actual data type can have a resolution less than or equal to TIMESTAMP. If the resolution is greater than TIMESTAMP (for example, nanosecond vs. microsecond), you may get occasional cases of missing or duplicate records.	Required
Timestamp column initial value	TimestampColumnInitValue	Timestamp	If this field is set, the first time that the query is executed, the adapter retrieves only records whose timestamp field contain values greater than the value specified in this Initial Value field. Data values equal to the Initial Value are not be retrieved. The format of the date should be "YYYY-MM-DD HH24:MI:SS.FF", where "FF" indicates microseconds (up to 6 digits).	Optional

Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

The Database Name property indicates which database to connect to. Despite the fact that we call this property the Database Name, it is actually the name of a service thatmust be configured in the `c8-services.xml` file. The service name is not necessarily identical to the name of the database. Each service has information about how to connect to the external database server and the name of the database on that server. For more information about configuring a service to provide access to a database on an external database server, see the *Sybase CEP Installation Guide* .

The database query is constructed from the Table Name, the Where clause, and the Timestamp Field. The query selects the columns of the table (or view) that match the names and data types of the fields in the target stream. This means that the stream schema must either match the table schema, or be a subset of the table schema.

Specify which field in the table should be used as the Timestamp field. The Timestamp field acts like the Row Timestamp in a file read by a Read From CSV File adapter. The Row Timestamp controls the order and relative timing with which rows are processed. The adapter uses the Timestamp field as the ORDER BY clause in the query. Once the rows arrive at the server, the server treats the rows as though they arrived at the same relative times as the Timestamp field (Row Timestamp) specifies. For example, if the row timestamps are 10,000 microseconds apart, then the server treats the rows as though they had arrived 10,000 microseconds apart. This Timestamp field can be one of the columns of the table; the Timestamp field can also be, but is not required to be, one of the columns in the stream schema.

The Loop Count property allows you to tell the adapter to read the same data more than once. This is useful in prototyping if you want to do a test run with a large amount of data but you only have a small amount of data available.

When looping is used, the row timestamps in the data are incremented appropriately for each loop so that time does not appear to jump backwards when you finish one loop and start the next (which of course involves re-reading rows you have already read).

For each loop iteration, the adapter re-runs the query. This allows the adapter to avoid buffering old query results, which would significantly hamper performance for large result sets.

Note: Since the query is re-executed each time, and since the table data can change between queries, the data may change between loop iterations.

When using looping, you can set property values to specify whether you want the entire set of records read on the first cycle, or whether you want to read only records that have a Timestamp field value greater than a value that you specify in the Timestamp Field Initial Value.

To read the entire table on the first cycle, leave Timestamp Field Initial Value empty. To read just the records with Timestamp field values greater than a specified value (for example, greater than 2007-09-01 11:30:00.0), put that specified value into the Timestamp Field Initial Value property.

Database: Write to DB Output Adapter

The Write to Database adapter writes information to an external database server.

This adapter is an in-process adapter. The Server starts it automatically when necessary. Attach this adapter to a stream by using commands inside Sybase CEP Studio. Properties for this adapter are set using Sybase CEP Studio.

The adapter properties are listed in the table below.

Property Name (Attach Adapter)	Type	Description
Query	String	Required. The query (fo

Property Name (Attach Adapter)	Type	Description
		r e x a m p l e , I N S E R T s t a t e

Property Name (Attach Adapter)	Type	Description
		ment) that you want to

Property Name (Attach Adapter)	Type	Description
		run on the external dat

Property Name (Attach Adapter)	Type	Description
		base server. This stat

Property Name (Attach Adapter)	Type	Description
		ement must be written i

Property Name (Attach Adapter)	Type	Description
		n t h e l a n g u a g e (f o r e x a

Property Name (Attach Adapter)	Type	Description
		mple, SQL) of the exter

Property Name (Attach Adapter)	Type	Description
		nal database server, no

Property Name (Attach Adapter)	Type	Description
		tin CCL. For example,

Property Name (Attach Adapter)	Type	Description
		do not use a CCL Databa

Property Name (Attach Adapter)	Type	Description
		se Statement (" EXECUTE

Property Name (Attach Adapter)	Type	Description
		STATEMENT DATABASE...")

Property Name (Attach Adapter)	Type	Description
		as the query.

Property Name (Attach Adapter)	Type	Description
DBName	String	Required. Database name

Property Name (Attach Adapter)	Type	Description
		. This must be the name

Property Name (Attach Adapter)	Type	Description
		of a service defined i

Property Name (Attach Adapter)	Type	Description
		n t h e c8-services.xml

Property Name (Attach Adapter)	Type	Description
		file. The service infor

Property Name (Attach Adapter)	Type	Description
		mation includes informa

Property Name (Attach Adapter)	Type	Description
		tion about how to conne

Property Name (Attach Adapter)	Type	Description
		ct to the external data

Property Name (Attach Adapter)	Type	Description
		base server and the nam

Property Name (Attach Adapter)	Type	Description
		e o f t h e d a t a b a s e t o c o

Property Name (Attach Adapter)	Type	Description
		nnect to.

Property Name (Attach Adapter)	Type	Description
CommitFrequency	Integer	Optional. If this field

Property Name (Attach Adapter)	Type	Description
		is left empty or is se

Property Name (Attach Adapter)	Type	Description
		t t o 0 , S y b a s e C E P S e r v

Property Name (Attach Adapter)	Type	Description
		er sends a commit for e

Property Name (Attach Adapter)	Type	Description
		a c h r o w s e n t t o t h e e x t

Property Name (Attach Adapter)	Type	Description
		ernal database server.

Property Name (Attach Adapter)	Type	Description
		If you want to commit e

Property Name (Attach Adapter)	Type	Description
		very N rows (for exampl

Property Name (Attach Adapter)	Type	Description
		e, every 3rd row) rathe

Property Name (Attach Adapter)	Type	Description
		than every row, then

Property Name (Attach Adapter)	Type	Description
		enter the desired value

Property Name (Attach Adapter)	Type	Description
		of N (for example, 3)

Property Name (Attach Adapter)	Type	Description
		into this field. See mo

Property Name (Attach Adapter)	Type	Description
		reinformation about Co

Property Name (Attach Adapter)	Type	Description
		mmit Frequency below.

The query is usually an INSERT statement that inserts values into the external database server. The INSERT statement typically contains placeholders for values; when a new row arrives in

the stream associated with this output adapter, values in that stream are substituted for the placeholders in the INSERT statement.

For example, the query might look like:

```
INSERT INTO table1 (col1, col2)
VALUES (?streamColA, ?streamColB);
```

If the Write To DB output adapter is on a stream named StockStream, then when a new row arrives in StockStream, the values from streamColA and StreamColB in StockStream will be substituted for the query properties "?streamColA" and "?streamColB", and the query will be sent to the external database server and executed there.

You can use statements other than INSERT. For example, you can use UPDATE or DELETE. You can also call stored procedures or other statements on the external database server. You could even use a SELECT statement as the query; however, since this is an output adapter, no values from the external server are returned, and thus a SELECT statement probably would not do anything useful. Similarly, calling a function whose sole effect is to return a value probably would not do anything useful, either.

The database name property must be the name of a service configured in the `c8-services.xml` file. For more information about configuring a service to provide access to a database on an external database server, see the Sybase CEP *Installation Guide* .

See *Datatype Mappings* on page 347 for tables that show which database data types correspond to Sybase CEP's data types.

More Information about Commit Frequency

If you set the commit frequency to a value larger than 0, then the adapter performs commit operations either when the specified number of rows have been sent, or when the queue of rows to write to the database is empty. This prevents ready-to-be-committed rows from having to wait indefinitely until the number of rows to be committed reaches N.

Warning! Uncommitted rows are not committed when a project is stopped. For example, suppose that you set the commit frequency to 5 minutes. As the project runs, it writes to the database every five minutes. However, when the project is stopped, a final commit is not executed. If the project stops two minutes after the most recent commit, then the most recent two minutes of database writes are not committed.

Email: Send Email Out Adapter (SMTP)

The Send Email Out adapter sends email messages with fixed "From", "To", "Cc", "Bcc", "Subject", and "Body" fields in every email message as configured in the Adapter.

For each message that this output adapter receives, this adapter sends email to the specified address(es).

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
SMTP Service	Service	String	Name of SMTPService definition in `c8-services.xml`.
To	To	String	Email address(es) to send the message to.
From	From	String	Email address that the generated email will be from.
Cc	Cc	String	Email address(es) to send the copy to.
Bcc	Bcc	String	Email address(es) to send the blind copy to.
Subject	Subject	String	Email subject.
Body template	BodyTemplate	String	Body template; in other words, the text of the message that you would like to send.
HTML Mail	HtmlMail	Boolean	Set this to true (in other words, put a checkmark in the box by clicking on the box) if you want the email sent in HTML format. Set this to false (in other words, remove the checkmark in the box by clicking on the box) if you do not want the email sent in HTML format.

Instead of specifying a specific value for the "From", "To", "Cc", "Bcc", "Subject", and "Body template" fields, you may use a property that tells the server to read the value from the stream. For example, suppose that your stream has a column named "ErrorCode", which contains an error number that you would like to include in the "Subject" field of the email message. You can put "ERROR = ?ErrorCode" in the "Subject" property of this adapter; when the recipient gets the message, the "?ErrorCode" will be replaced with the value in the ErrorCode field of the stream. The question mark character ("?") immediately preceding the "ErrorCode" tells

the adapter to get the value in the ErrorCode field, rather than use "ErrorCode" as a literal value.

The `c8-services.xml` file needs to have an entry like this:

```
<Service Name="MySMTPServiceName" Type="SMTP">
  <Description>Description of MySMTPService</Description>
  <!-- Hostname of SMTIP service. Required -->
  <Param Name="Server">smtp.server.hostname</Param>
  <!-- Port number of SMTP service. Optional -->
  <Param Name="Port"></Param>
  <!-- Username for SMTP service access. Optional -->
  <Param Name="Username">smtp.server.username</Param>
  <!-- Password for SMTP service access. Optional -->
  <Param Name="Password">smtp.server.password</Param>
</Service>
```

Obviously, the name here needs to match the name on the SMTPService property on the adapter. You need to correctly configure the Server param and optionally the Username and Password.

Java Email Output Adapter

The Java Email Output adapter sends email to the specified recipient(s). When a row is received by this Adapter, the Row information is put into the body of an email message and then sent out.

The Java Email Output adapter is an out-of-process adapter.

Set your CLASSPATH environment variable according to the instructions in *Setting Up Your Environment* on page 279. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by or written by you.

To run this adapter, execute the following command:

```
java com.sybase.c8.adapter.EmailOutputAdapter <parameters>
```

Each parameter is of the form

```
--<paramname>=<paramvalue>
```

Note that "--" (two dashes) must precede each parameter name.

For example, the beginning of such a command line looks similar to the following:

```
java com.sybase.c8.adapter.JavaEmailAdapter --url= ...
```

Use the command-line parameters shown in the table below. These parameters are case-sensitive.

Parameter	Description	Required?
url	The CCL URL of the stream to read from.	Required
smtpHost	This is the name of the SMTP host.	Required
from	The email address that the email message will show the email as being from.	Required
to	The email address(es) to send the messages to.	Required
subject	The email subject.	Optional
body	The body of the message. (Note that you may use either the "--body" or the "--bodyFile" option, but not both.)	Optional
bodyFile	A file that contains the body of the message. (Note that you may use either the "--body" or the "--bodyFile" option, but not both.)	Optional
smtpPort	The port that the SMTP program uses to communicate.	Optional
smtpUsername	The user name.	Optional
smtpPassword	The password.	Optional
useTLS	Use a TLS secure connection when connecting to the mail server. This option works only if your mail server supports a TLS connection. You can use either the "--TLS" option or the "--SSL" option, but not both.	Optional

Parameter	Description	Required?
useSSL	Use an SSL connection when connecting to the mail server. This option works only if your mail server supports an SSL connection. You can use either the "--TLS" option or the "--SSL" option, but not both.	Optional
timeout	SMTP timeout in milliseconds.	Optional
timeFormat	The date format in the format of java.text.SimpleDateFormat.	Optional

It is possible to insert values from the stream into the subject and body of the message. Do this by including variables in the subject or body. A variable is the name of a field, with a question mark immediately preceding that field. For example, if you want to insert the value of the field named Price into the body of the message, then put ?Price in the body. Your body might look like:

```
The price reached ?Price.
```

It is also possible to insert values from the stream into the "to" field -- in other words, to specify to whom the message should be sent. Below is an example command line that starts the server and specifies that the "to" field should come from the stream:

```
java com.sybase.c8.adapter.EMailOutputAdapter ... --to=?
ContactField ...
```

The message will be sent to the person whose email address is specified in the "ContactField" column of the stream.

SSL uses a secure connection from the beginning, while TLS is a method provided by some servers to allow an unencrypted connection to connect and negotiate an encrypted connection for the remainder of that session a port that may also allow non-secure traffic. Depending on how the server is configured, it may require that the session be escalated to an encrypted connection before anything of importance can be done over it. This method allows a mail server to use a single port for both non-encrypted and encrypted traffic, rather than requiring separate ports for encrypted and non-encrypted traffic.

If you do not specify either the -body or the -bodyfile option, then the body of the email message will contain the XML representation of the Sybase CEP row/message that arrived at the adapter.

Ganglia Input Adapter

The Ganglia adapter takes data from a Ganglia Monitoring System source and sends the rows to the Sybase CEP stream.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
MCAST Port	MCastPort	Integer	The Ganglia MCAST port. This property is required. The default value is 8649.
MCAST Group	MCastGroup	String	The Ganglia MCAST group. This property is required. The default value is 239.2.11.71.
Resolve hostnames	ResolveHostnames	Boolean	If this is set to true, the adapter attempts to convert IP addresses into real hostnames by doing a reverse lookup.

Each input stream has a property that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

For more information see the Ganglia Alerts example in the examples/NetworkAndSecurity/GangliaAlerts directory.

On Microsoft Windows, the directory is typically:

```
C:\Program Files\SybaseC8\Server\examples\NetworkAndSecurity
\GangliaAlerts
```

On UNIX-like operating systems, the directory is typically:

```
/home/<username>/sybasec8/server/examples/NetworkAndSecurity/
GangliaAlerts
```

JDBC Input Adapter

The JDBC Input adapter attempts to establish a JDBC session with the specified database. It then executes the given query periodically based on the specified frequency.

Each time the query is submitted, the data stream gets rows that have columns with names and types equivalent to those selected from the source table. The SQL types that map to CCL data types are shown in the Datatype Mappings table in *JDBC Output Adapter* on page 473 .

This adapter is run as an out-of-process adapter.

Set your CLASSPATH environment variable according to the instructions in *Sybase CEP Java SDK* on page 279. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Sybase CEP or written by you.

Execute the following command:

```
java com.sybase.c8.adapter.JDBCInputAdapter <parameters>
```

Each parameter is of the form

--<paramname>=<paramvalue>

For example, the beginning of such a command line will look similar to the following:

```
java com.sybase.c8.adapter.JDBCInputAdapter --queryFile=q1.sql ...
```

Note that "--" (two dashes) must precede each parameter name.

Use the command-line parameters shown in the table below. These parameters are case-sensitive.

Parameter	Description	Required/Optional
queryFile	A file with the SQL query, which can be almost any SQL statement that is valid for the database that the adapter is connected to. The query can be parameterized.	Required
username	The username with which to log into the database.	Required
password	The password for the specified user name.	Required
databaseUrl	A database address in the standard form used by JDBC. For example, --databaseUrl=jdbc:oracle:oci8:@oracle.	Required
streamUrl	The URL of the stream to send tuples to.	Required

Parameter	Description	Required/Optional
driver	The database driver to use. For example: --driver=oracle.jdbc.driver.OracleDrive. Sybase CEP does not bundle JDBC drivers with the product, so customers must provide the drivers themselves. In order for the Sybase CEP JDBC Input adapter to work, the JDBC driver must be included in the Java CLASSPATH, as well as pass the driver on the command line.	Required
pollInterval	Frequency in microseconds with which to periodically resubmit the query.	Required
tupleDescriptorFile	The name of a file with a tuple descriptor that describes the schema that you are publishing to. See the discussion of the tupleDescriptor File below.	Required
fetchSize	The size of the batch to read from the database and accumulate before sending tuples to the stream. (The default is 10.)	Optional
incrementalUpdates	If this is set to true, the adapter adds the retrieved rows to the existing rows. If this is set to false, the adapter replaces all the rows each time that the query is executed. Set this to true if you are repeatedly retrieving the same relatively small, semi-static data set.	Optional

Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

After the adapter starts, it initializes its connection to Sybase CEP Server and the remote database server. The adapter then sleeps the initial hard-coded time and then starts executing the query against the database.

To shut down the adapter, send an interrupt signal (for example, ctrl-C).

Like any out-of-process adapter, this adapter requires that the server and query module already be running, and that the stream exists so that the adapter can attach to it.

Installing and configuring Oracle JDBC adapters

Describes the requirements for installing Oracle JDBC adapters, including installing the Oracle client, and the command to call the adapter.

We use "thick" drivers in our JDBC adapters. Therefore, you need to install Oracle client on the computer where adapters will run. You can use Oracle's InstantClient, which is lightweight, and has everything that you need. After you have installed this, modify your CLASSPATH to include the 2 .jar files that came with the client: classes12.jar and odbc14.jar Modify your LD_LIBRARY_PATH to point to the directory with the libraries from the client.

Set up tnsnames.ora (this is optional). This is purely for convenience. If you set up tnsnames, you can call the database by its alias in the command line for the adapter. Otherwise you'll have to provide the full URL (see below)

Here's a sample command to call JDBCInputAdapter to connect to an Oracle database (no tnsnames used):

```
java com.sybase.c8.adapter.JDBCInputAdapter --queryFile=query.sql
  --tupleDescriptorFile=td2.xml --username=lola
  --password=sybase --driver=oracle.jdbc.driver.OracleDriver
  --databaseURL=jdbc:oracle:oci8:@oraclehost.sybase.com:1521:orcl
  --pollInterval=30000
  --streamUrl=ccl://ab03.sybase.com:7777/Stream/PQA/DbWrite/
InStream
```

Tuple Descriptor File

The Tuple Descriptor File must describe the schema of the stream that you are publishing to.

This file is required because out-of-process adapters have no way of knowing the schema on Sybase CEP Server side unless we tell it to them. An example tuple descriptor looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<TupleDescriptor
      xmlns="http://www.sybase.com/cpx/2004/03/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.sybase.com/cpx/2004/03/
cpx_td.xsd"
      Name="MyTuple"
      >
   <Field Name="field1" xsi:type="StringFieldType" />
   <Field Name="field2" xsi:type="IntegerFieldType" />
</TupleDescriptor>
```

JDBC Output Adapter

Provides information about configuring and connecting with the JDBC Output Adapter.

Configuring the JDBC Output adapter

A JDBC Output adapter enables you to write output to another database server through JDBC.

In order to use this adapter you must specify several pieces of information, including:

- Information about connecting to the database server. This could be a connection string or other connection information, a user ID with appropriate privileges on the external database, and a password.
- An SQL statement (typically an INSERT statement) that inserts a row into the external database. This statement contains placeholders for parameters; the values for the parameters are read from each row received by the output adapter.

After the JDBC Output Adapter is loaded, it attempts to establish a JDBC session with the given database. It then executes the specified SQL statement on each row it receives from the output data stream. The query is parameterized in the fashion of a JDBC PREPARE statement. The columns in each row are inserted into the query in the order they appear in the row descriptor. See JDBC standard documentation for details

The JDBC Output adapter submits a database INSERT query in order to write each row of output. The parameters necessary to connect to the external database are specified in theSybase CEP Server configuration file. For more information about Sybase CEP Server configuration file, see the *Sybase CEP Installation Guide* . Whenever a row is output to this Adapter, the row is INSERTed into the table specified in the input query statement.

This adapter is run as an out-of-process adapter. To run it, execute the following steps:

1. Set your CLASSPATH environment variable according to the instructions in *Java SDK* on page 279. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Sybase CEP or written by you.
2. Execute the following command:

```
java com.sybase.c8.adapter.JDBCOutputAdapter <parameters>
```

Each parameter is of the form

--<paramname>=<paramvalue>

For example, the beginning of such a command line will look similar to the following:

```
java com.sybase.c8.adapter.JDBCInputAdapter --queryFile=q1.sql ...
```

Note that two dashes must precede each parameter name.

Use the command-line parameters shown in the table below. These parameters are case-sensitive.

Parameter	Type	Required/Optional
queryFile	A file with the SQL query, which can be almost any SQL statement that is valid for the database that the adapter is connected to. The query can be parameterized.	Required
username	The username with which to log into the database.	Required
password	The password for the specified user name.	Required
databaseUrl	A database address in the standard form used by JDBC. For example, --databaseUrl=jdbc:oracle:oci8:@oracle.	Required
streamUrl	The URL of the stream to read tuples from.	Required
driver	The database driver to use. For example: --driver=oracle.jdbc.driver.OracleDrive. Sybase CEP does not bundle JDBC drivers with the product, so customers must provide the drivers themselves. For the Sybase CEP environment, you can use the Oracle driver with the Oracle product. In order for the Sybase CEP JDBC Input adapter to work, the JDBC driver must be included in the Java CLASSPATH, as well as pass the driver on the command line.	Required

Here's a sample command to call JDBCOutputAdapter to connect to a database (no tnsnames used):

```
java com.sybase.c8.adapter.JDBCOutputAdapter
--queryFile=/home/lita/QueryFileOutputAdapter --username=lita
--password=SYBASE
--databaseURL=jdbc:oracle:oci8:@oracle.eng.sybase.com:1521:orcl
--driver=oracle.jdbc.driver.OracleDriver
--streamUrl=ccl://pc03.sybase.com:7777/Stream/DbWrite/OutStream
```

When a value from the Sybase CEP output data stream is inserted into an external database, the data type of the Sybase CEP output value must be compatible with the data type of the corresponding column in the table in the external database. The table below shows the mapping between CCL data types and SQL data types. It is assumed that the output table exists in the destination database and has types that conform to the ones shown.

Datatype Mappings between CCL and ANSI SQL

CCL Type	ANSI SQL Type	Description
INTEGER	INTEGER	Integer values between -2147483648 and +2147483647 (-2^31 to +2^31 - 1).
LONG	(See footnote 1)	Integer values between -9223372036854775808 and +9223372036854775807 (-2^63 to +2^63-1).
FLOAT	FLOAT	64-bit floating point numbers.
STRING	CHARACTER VARYING(2147483647)	Character strings.
TIMESTAMP	TIMESTAMP	Date and time, specified as years, months, days, hours, minutes, seconds, and fractions of a second.
INTERVAL	INTERVAL DAY TO SECOND (See footnote 2)	Interval of time, specified as days, hours, minutes, seconds, and fractions of a second.

Footnotes:

1. The LONG data type is not part of the ANSI SQL-92 specification.
2. Sybase CEP does not support YEAR TO MONTH intervals.

   Your query must use the same number of binding variables as there are fields in the stream schema that you use. For example, if a stream that feeds the output adapter has 3 fields in it, the query must have 3 binding variables. Since this is an unmanaged adapter, it cycles through each row received and tries to place each column in the row in a corresponding binding variable.

   Like any out-of-process adapter, this adapter requires that the server and query module already be running, and that the streams exist so the adapter can attach to them.

   To shut down the adapter, send an interrupt signal (for example, ctrl-C).

Connecting to a remote database server with the JDBC Output adapter

A sample command to call JDBCOutputAdapter to connect to a remote database server.

No Oracle tnsnames are used:

```
java com.sybase.c8.adapter.JDBCOutputAdapter
 --queryFile=/home/lita/QueryFileOutputAdapter
 --username=lita --password=SYBASE
 --databaseURL=jdbc:oracle:oci8:@oracle.eng.sybase.com:1521:orcl
 --driver=oracle.jdbc.driver.OracleDriver
 --streamUrl=ccl://pc03.sybase.com:7777/Stream/OutStream
```

For more information about connecting on UNIX-like operating systems, see *Connectivity Instructions for UNIX-like Operating Systems* on page 472.

JMS Adapter

The JMS Adapter is an out-of-process adapter. You can configure the JMS Adapter as either an input adapter or an output adapter. Both adapters work with JMS messages that have message body type "MapMessage" or "TextMessage".

This adapter attempts to establish a JMS (Java Messaging Service) session and does one of the following:

- If the adapter is configured as an input adapter, it subscribes to the given topic and listens on that session. When the adapter receives a message from the JMS server, it sends the message as a tuple into the Sybase CEP Server (in other words, into a stream), framed by reset and commit tuples. Messages must be set up in such a way that fields in MapMessages correspond exactly to the stream's tuple descriptor.
- If the adapter is configured as an output adapter, it takes data received from a Sybase CEP Server stream and publishes the data to the JMS topic. Messages sent out are JMS MapMessages. The order and type of fields in the JMS messages must exactly match order and type of fields in the Sybase CEP stream's tuple descriptor.

Note: In order to use the Sybase CEP Adapter for JMS to integrate or communicate with TIBCO Enterprise Message Services, you must have a valid license for TIBCO Enterprise Message Services from TIBCO or from an authorized TIBCO channel.

The JMS server should use the JMS 1.1 protocol.

This adapter can be configured to use Sybase CEP's Guaranteed Delivery when reading from or writing to a Sybase CEP stream. If GD is turned on, then the adapter provides At Least Once semantics for outbound MapMessage and TextMessage messages from Sybase CEP and Exactly Once semantics for inbound messages.

To run the JMS adapter, execute the following steps:

1. Make sure that you have a JMS server up and running.
2. Make sure that you have installed the JMS client.

3. Set the desired values in the jndi.properties file, and set your CLASSPATH to include the directory that holds the jndi.properties file. The CLASSPATH should include only the path, not the path plus the filename, of the jndi.properties file.
4. Set your CLASSPATH environment variable according to the instructions in *Sybase CEP Java SDK* on page 279. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Sybase CEP or written by you.
5. Update your CLASSPATH to include the necessary JMS libraries.

   Note: Sybase CEP does not supply the libraries for any JMS implementation.
6. Execute the following command:

   ```
   java com/sybase/c8/adapter/JMSAdapter <parameters>
   ```

   Each parameter is of the form
   --<paramname>=<paramvalue>
   note that "--" (two dashes) must precede each parameter name. For example:

   ```
   java com/sybase/c8/adapter/JMSAdapter --topic=test1 --GD=true ...
   ```

   Use the command-line parameters shown in the table below. These parameters are case-sensitive.

Parameter	Description	Required/Optional
kind={input | output}	Indicates whether the adapter should run as an input adapter or an output adapter.	Required.
serverURL=<URL to server>	URL of Sybase CEP Server, for example `http://localhost:6789` . If your manager and containers are separate, this must be the URL of the manager.	Required.
queue=<queueName>	Name of the JMS queue, to be looked up in JNDI.	You must specify either a queue or a topic.
topic=<topicName>	Name of the JMS topic, to be looked up in JNDI.	You must specify either a queue or a topic.
GD={true | false}	Indicates whether to use Guaranteed Delivery. The default value is false. For best results, the stream's GD setting should match the adapter's setting.	Optional.

Parameter	Description	Required/Optional
sessionID=<text>	The session ID is an arbitrary identifier, but must be unique among all publishers to a stream and among all subscribers to a stream. An adapter instance must be started with the same sessionID each time.	Required for GD.
clientID=<text>	A JMS clientID.	Required for GD input using a topic.
durableName=<text>	The name used within the clientID for the durable subscription (JMS GD input adapter only).	Required for GD input using a topic.
factoryName=<factoryName>	The name of the factory in JMS. This varies depending on the which vendor's JMS you are using.	Required.
initialNamingFactory= <initialNamingFactory>	The initial naming factory for JMS implementation. For example, the Joram JMS implementation uses `fr.dyade.aaa.jndi2.client.NamingContextFactory` as the value. This value is also the default value.	Optional.
streamURL=<CCL_URL>	The CCL URL of the stream to which the adapter should publish or subscribe. For example, `ccl://localhost:6789/Stream/Default/JMSTest/InStream`	Required.
messageType=<message type>	MapMessage or TextMessage. The default is MapMessage.	Optional.
C8username=<userID>	If this is set, then the connection to Sybase CEP Server uses credentials to authenticate the user.	Optional.
C8password=<password>	The password for the specified username, used in the credentials along with the c8username.	Optional.

Parameter	Description	Required/Optional
JMShost=<hostname>	The host computer on which the JMS server is running. The default value is `localhost` .	Optional.
JMSport=<port>	The port that the JMS server is using. The default value is 16400.	Optional.
batchSize=<integer>	If the adapter is using GD, then the adapter will attempt to send rows in batches of the specified size. When running as an input adapter, the adapter sends the data to the Sybase CEP Server as soon as either the batch size or commitIntervalMsec is reached, whichever comes first. If both batchsize and rate are set, rate is used. When running as an output adapter, the adapter performs a JMS transaction when it reaches either the batchSize or the commitIntervalMsec, whichever comes first. The default batch size is 500.	Optional.
rate=<float>	For GD input, the adapter attempts to send at approximately this rate. If both rate and batchSize are set, rate is used.	Optional.
commitIntervalMsec=<integer>	The adapter sends a batch at least this often (in milliseconds) unless there are Optional. rows to send. The default value is 500 milliseconds.	Optional.

Parameter	Description	Required/Optional
statusFile=<filename>	The file name to which the adapter should write status if the adapter is either a GD output adapter, or aGD input adapter for topics. File names must be different for each instance of the adapter. The default file name is `JMSA-dapter.status.`	Optional.
debug=true	Indicates whether to print out debug information about "high-level" operations. By default, debugging is off.	Optional.
debugVerbose=true	Indicates whether to print out debug information about each message. By default verbose debugging is off.	Optional.

Although the GD settings of the stream and adapter are not required to match, we strongly recommend that both turn on GD or both turn off GD. A stream's GD settings may be specified at the level of the stream, or can be inherited from the project that the stream is within.

When the JMS adapter constructs the message, the adapter uses the current time as the message time. This can cause issues if the commit interval for the server is larger than the max delay (or even approximately the same length as the max delay), or if the adapter machine's clock is set significantly later than the server machine's clock.

Troubleshooting: When the adapter starts, it prints all the arguments set on the command line and in the `jndi.properties` file. Check these values to ensure that the arguments the adapter used are the arguments you intended. The `jndi.properties` printing serves two purposes. First, it shows that the adapter found a jndi.properties file in one of the directories specified in the CLASSPATH. Second, it shows the contents.

If you use GD with topics as opposed to queues, you will need durable subscriptions. Not all JMS implementations support durable subscriptions.

Random Tuples Generator Adapter

The Random Tuples Generator adapter –sometimes called the Random Messages Generator – is an input-only adapter that generates random tuples according to the given schema and sends the rows to the stream.

The Random Tuples Generator Adapter is primarily used for prototyping and basic testing. Both its schema and configuration file are user-editable.

The following configurable parameters are defined in `RandomTupleGenInAdapter.xml`:

Property Name (Screen)	Property Name (Attach Adapter)	Type	Description
Rate	Rate	Float MIN: 0 MAX: 1,000,000 Default: 100	Optional. The number of Rows generated per second. If the rate is negative, the adapter will stop and return a fatal error message to the server. If the rate is larger than the MAX value, it will reset to the MAX value and report a warning message to the server. If the rate parameter is blank, it will reset to the Default value and report an info message.
Row Count	RowCount	Long MIN: 0 MAX: 2,000,000,000 DEFAULT: 0 (infinity)	Optional. How many rows to generate. If RowCount is negative, the adapter will stop and return a fatal error message to the server. If RowCount is larger than the MAX value, it will reset to the MAX value and report a warning message to the server. If the RowCount parameter is blank, it will reset to the default value and report an info message to the server.

Property Name (Screen)	Property Name (Attach Adapter)	Type	Description
Timestamp Base	TimestampBase	Current time	Optional. The initial time for message timestamps. The supported format of Timestamp is YYYY-MM-DD HH24:MI:SS:FF If TimestampBase is blank, it will reset to the default value. If TimestampBase has an invalid timestamp format, the adapter will stop and return a fatal error message to the server.

Data generated by this adapter is not evenly distributed across the range of possible values for each data type. The following table shows the value range generated for each data type:

Data Type	Range of Values
Bool	true/false
Int	0 .. 99 inclusive
Long	0 .. 99 inclusive
Float	0.0 .. 10.0 exclusive (should never get 10.0)
Interval	0 .. 9 inclusive
Timestamp	Sets value to current time
CharPtr	2 characters from the following ranges a..z, A..Z, 0..9
BlobPtr	2 bytes each with range 0..255

Note: Even within this range, values are not necessarily evenly distributed.

RAP - The Trading Edition Adapters

RAP - The Trading Edition™ writes in-bound market data to an in-memory cache database and a historical data store. Sybase CEP provides in-process and out-of-process RAP adapters to facilitate communication between the two products. This document assumes that you have installed RAP - The Traditing Edition ™ and are familiar with its components.

RAP - The Trading Edition In-process Adapter

The RAP - The Trading Edition in-process adapter reads a Sybase CEP data stream and transforms it into a feed handler for RAP.

The Sybase CEP RAP in-process adapter, named `c8_write_to_rap.adl`, is located in the plugins folder of your Sybase CEP Studio installation directory. The file exists in the server/plugins directory as well. The adapter uses C/C++ and supports only Linux and Solaris platforms.

In Sybase CEP, container processes run queries and C/C++ adapters; each container in a Sybase CEP cluster also stores message templates for all applicable adapters. Any changes or additions you make to message templates must be reflected in every container in a cluster, and require you to restart every container. Use the RAP in-process adapter in a Sybase CEP cluster with only one active container, or one with a small number of containers that you can process together, to avoid disruptions on a large scale.

The `c8_write_to_rap.adl` file is located in the plugins folder of your Studio installation directory. Before you run the RAP in-process adapter, define the following parameters in the ADL file:

- RAPMessageType - (required) specifies which tables in RAP to send messages to based on message type. The RAPMessageType must be the same as the MessageType in the RAP data stream (RDS) Template.
- CEPTimeStamp - (optional) specifies the Sybase CEP Time Stamp at the beginning of each message.

The RAP in-process adapter must be initialized before messages are recieved from the Sybase CEP Server. In the Sybase CEP Studio Project menu, attach the adapter to an output stream by selecting the adapter from a drop-down list of output adapters. Support multiple publishers and subscribers by attaching the adapter to multiple streams. The RAP in-process adapter communicates with RAP - The Trading Edition through its RAP Publisher API.

RAP - The Trading Edition Out-of-process Adapter

The RAP - The Trading Edition out-of-process adapter reads a Sybase CEP data stream and transforms it into a feed handler for RAP.

This out-of-process adapter, named `c8_sybase_rap_adapter`, is installed in the adapters directory under your Sybase CEP Server or Sybase CEP Studio installation directory. The file exists under the following locations:

- SybaseC8/server/bin
- SybaseC8/server/adapters/rap/bin
- SybaseC8/enterprise-adapters/rap-pub/bin

The adapter uses C/C++ and supports only Linux and Solaris platforms.

The RAP adapter takes a single parameter that identifies a file containing preferences information. The file `sample_sybase_rap_prefs.xml`, also located in the adapters

directory, contains thorough comments explaining the file structure and the purpose of each preference. You can edit a copy of the sample preferences file to meet your requirements.

Before you run the adapter, be sure to modify your PATH and LD_LIBRARY_PATH environment variables to include the bin directory under your Sybase CEP Server or Sybase CEP Studio installation directory.

To run the adapter, execute the following, replacing `sample_sybase_rap_prefs.xml` , located at SybaseC8/enterprise-adapters/rap-pub/rap-r2/sample/, with the name of your preferences file:

```
c8_sybase_rap_adapter my_prefs_file.xml
```

Setting up UA for RAP - The Trading Edition Adapters

Unified Agent (UA) is a Java based agent that gathers statistics from a RAP publisher.

It uses JMX technology and runs parallel to the server it monitors or administers. Sybase recommends that you copy the `bin/rap/ua` directory and use a different `ua` directory for each running instance of a RAP adapter.

To start the UA agent and configure the RAP in-process or out-of-process adapter to package UA statistics:

1. In `bin/rap/ua/bin/myenv`, set appropriate values for the following envars as specified in the "TODO" instructions at the top of the file:
   - SYBASE_UA
   - SYBASE_JRE
   - UA_RMI_PORT

   Note: UA_RMI_PORT in myenv should match the port property value in `service-config.xml`.
2. In `bin/rap/ua/plugins/com.sybase.rap.rapservice/agent-plugin.xml`, set appropriate values for the following properties as specified in the file:
   - rap.service_host
   - rap.service_port
   - rap.service_startCommand
   - rap.service_pluginsubtype
   - rap.service_plugindisplayname

   If the `bin/rap` directory link points to the `rap-r4` directory, set the following properties in `agent-plugin.xml` also:
   - rap.service_logFile
   - rap.service_configFileDir
   - rap.service_componentTemplateDir
3. In `in/rap/ua/service/RMI/service-config.xml`, set the appropriate value for the following property as specified in the file:

- port

Note: The AdminPort node value in the adapter `publisher.xml` should match the rap.service_port property value in `agent-plugin.xml`.

Working with Multiple Versions of RAP - The Trading Edition

This section reviews how to integrate Sybase CEP with RAP - The Trading Edition, versions R3 and R4.

Sybase CEP provides two wrapper libraries (`libc8_sybase_rap_publisher_r2_lib.so` and `libc8_sybase_rap_publisher_r4_lib.so`) that you can use to interact with the RAP Publisher API of each release. The names of the wrapper libraries are versioned. The RAP in-process and out-of-process adapters link against a versionless library name.

Sybase CEP can only run against one version of RAP at a time. Depending on which RAP version you want to publish to, create a soft link between the versionless library and one of the versioned wrapper libraries.

```
libc8_rap_pub_lib.so -> libc8_sybase_rap_publisher_r2_lib.so
```

Or:

```
 libc8_rap_pub_lib.so -> libc8_sybase_rap_publisher_r4_lib.so
```

Regular Expressions: Read From File Using a Regular Expression Adapter.

The Read From File Using a Regular Expression adapter reads an input file line by line, matches each line against a given regular expression, and produces rows.

Only rows that match all of the subexpressions are inserted into the stream, and only the portions of those rows that match the subexpressions are inserted into the stream. (Effectively, you are selecting both rows and columns from the input.)

POSIX regular expression syntax is used. This is compatible with Perl-style regular expressions. Use "..." to denote sub-expressions. The order of sub-expressions must match the Row descriptor order. Binding of RegEx matches to fields is performed by position. The name is ignored.

For example, if you search using the following regular expression:

```
/(J).*(SMITH)/
```

and the input rows are:

```
1) JOHN SMITH
2) JANE SMITHERS
3) JANE AUSTEN
4) DAVE SMITH
5) JANET SMITHSONIAN
```

This returns three rows (taken from rows 1, 2, and 5 above), each with 2 columns of output:

```
J SMITH
J SMITH
J SMITH
```

More sophisticated patterns allow you to extract a broader range of values, such as the complete last names.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Filename	Filename	String	The name and path of the file to read data from. You may specify a file name and path either by typing it in or by clicking the Browse button for this field. The path is relative to the server's adapters base folder and must be underneath that base folder. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. For details about the Browse and Edit buttons to the right of the filename, see the discussion following this table.
Loop count	LoopCount	Integer Min: 0 Max: 2000000000 Default: 1	If the Loop count is 1, the file is read only once and the adapter stops sending data once the end of file is reached. If the Loop count is greater than 1, then after finishing reading the file, the adapter start readings again from the beginning. If the Loop Count is 0, the adapter repeats indefinitely.
Rate	Rate	Float	If this property is non-zero, then the adapter reads data from file at the given rate (per second). Any time-stamps in the input file are ignored.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Timestamp Base	TimestampBase	Timestamp	The point at which time starts for this adapter, relative to the file's first timestamp. For example, if Timestamp base is 0 and the first row's timestamp is 5000000 (in microseconds), the first row is sent 5 seconds (5000000 microseconds) after the module starts. If blank, the first row is sent immediately after the module starts.
Set Timestamp To Current Time	UseCurrentTimestamp	Boolean	If set to true, the adapter overrides the timestamp specified in the file with the current system time. Defaults to false.
Regular expression for parsing rows	RegEx	String	Regular expression to use when parsing rows.
Fields	Fields	String	Comma-separated list of fields corresponding to subexpressions.
Ignore Mismatch	IgnoreMismatch	Boolean	If true, a line not matching the regular expression is ignored. If false, a line that does not match the regular expression causes the adapter to raise an error condition and stops.
Log Mismatch	LogMismatch	Boolean	If true, lines that don't match the regular expression are logged. If false, the mismatched lines are dropped silently. This option is only used when Ignore Mismatch option is set to true.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Timestamp column format	TimestampColumnFormat	String	Specifies the format of the timestamp columns (for example, YYYY/MM/DD HH24:MI:SS.FF). If no timestamp format is specified, the adapter assumes that the timestamp is represented as a number of microseconds from 00:00:00 Jan 1, 1970 UTC/GMT. . Note that this format specifier applies to all input columns of type TIMESTAMP, not only to the row timestamp column. This means that ALL columns of type TIMESTAMP must be formatted the same way; you cannot specify independent formats for each TIMESTAMP column. For more information, see *Reading, Writing, and Converting Timestamps* on page 384

This adapter always has a 3-second startup delay before the first row is sent. If looping is used, the 3-second delay occurs only on the first iteration of the loop, not subsequent iterations.

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter

The "Browse" button for the filename property: The Adapter Properties screen in Sybase CEP Studio allows you to specify an input path and file name for the file by clicking the Browse button and identifying the file. The path is relative to the adapters base folder, either as specified for Sybase CEP Server running on the same computer as Sybase CEP Studio, or as specified in the preferences for Sybase CEP Studio. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. In order for this feature to work properly, make sure that the Adapter's Base Folder field in Studio Settings is set to the same folder as the adapters base folder for Sybase CEP Server. The Base Folder setting for the Server is specified during the installation process, and can also be changed later in the Sybase CEP Server's `c8-server.conf` file. The base folder setting for Sybase CEP Studio can be set from the Tools->Settings command on the Sybase CEP Studio menu.

The "Edit" button for the filename property: The edit button opens an editor that will allow you to edit any file whose name you entered into the filename field. This allows you to correct errors in the data. If the file's extension is "csv" or "xml", Sybase CEP Studio opens the appropriate editor specified in the "External Tools" tab available from the menu item "Tools -> Settings". For files with other extensions, on Microsoft Windows the editor is the one specified by the operating system's file associations. On UNIX-like operating systems, Sybase CEP Studio opens the editor specified by the EDITOR environment variable.

Note: When you click the Edit button, Sybase CEP Studio will look for the file in the Sybase CEP Repository, even if the adapters base folder is set to another location. You may need to use the Browse button (adjacent to the Edit button) to navigate to the desired directory before you try to edit the file.

Regular Expressions: Read From Socket Using a Regular Expression Adapter

A Read from Socket Using a Regular Expression adapter opens a TCP connection to a given address specified through Host and Port properties. Once a connection is established, the adapter reads tuples from this connection line by line, matching each line against a given regular expression and producing tuples from subexpressions.

POSIX regular expression syntax is used. Use "..." to denote sub-expressions. The order of sub-expressions must match the tuple descriptor order. See the description of the *Regular Expressions: Read From File Using a Regular Expression Adapter* on page 485 for more details about regular expressions.

If a connection is lost during adapter execution, the adapter attempts to reconnect.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Host	Host	String	The host name or IP address of the data source.
Port	Port	Integer Min: 1 Max: 65535	The port number of the data source.
Regular expression	RegEx	String	A regular expression to use when parsing rows.
Fields	Fields	String	A comma-separated list of fields corresponding to sub-expressions.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Timestamp column format	TimestampColumn-Format	String	The format of the timestamp columns (for example, YYYY/MM/DD HH24:MI:SS.FF). If no timestamp format is specified, the adapter assumes that the timestamp is represented as a number of microseconds from 00:00:00 Jan 1, 1970 UTC/GMT. Note that this format specifier applies to all input columns of type TIMESTAMP, not only to the row timestamp column. This means that ALL columns of type TIMESTAMP must be formatted the same way; you cannot specify independent formats for each TIMESTAMP column. For more information, see *Reading, Writing, and Converting Timestamps* on page 384.
Ignore Mismatch	IgnoreMismatch	Boolean	If true, a line not matching the regular expression is ignored. If false, a line that does not match the regular expression raises and the adapter is stopped.
Log Mismatch	LogMismatch	Boolean	If set to true, lines that don't match the regular expression are logged. If false, the mismatched lines are dropped silently. This option is only used when IgnoreMismatch option is set to true.

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter

RSS Feed Reader Adapter.

An RSS Feed Reader adapter opens a connection to a given URL and, once a connection is established, reads rows from this connection.

Property Name (Screen)	Property Name (Attach Adapter)	Type	Description
URL	URL	String	The RSS feed URL.
Ignore URL Errors	IgnoreErrors	Boolean	Specifies how the adapter behaves if it cannot connect to the URL when it starts. If true, the adapter attempts to reconnect every five minutes. If false, the adapter exits.
Refresh interval (microseconds)	RefreshInterval	String. Default: 60000000 microseconds (60 seconds)	Specifies how often to poll the RSS feed at the given URL. This can be a number of microseconds or a string of the form "# <unit>", for example, "10 seconds" or "1 minute". Note that not all valid formats for the INTERVAL data type may be used in this field.

Note: Each input stream has a property (see the stream's Properties tab in Sybase CEP Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

The adapter filters out the messages it has already seen and publishes to the stream only the messages that are new in the feed using the 'pubDate' RSS message field. The RSS feed item is converted to a message using mapping between Sybase CEP schema column names and RSS feed item elements. For example, if the Sybase CEP schema has 3 fields named title, link, and description, then the Sybase CEP message for the following RSS "item":

```
<item>
<title>Star City</title>
```

```
<link>http://nasa.gov/news/SpacePoll.asp</link>
<description>Space Exploration</description>
<pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
<guid>http://nasa.gov/2003/06/03.xml#item573</guid>
</item>
```

is:

```
"title", "link", "description"
--------------------------------
"Star City", "http://nasa.gov/news/SpacePoll.asp", "Space
Exploration"
```

Source code for this adapter is provided. If you wish to create a customized version of this adapter\:

- Copy and modify the files `c8_rss_reader.cpp` and `c8_rss_reader.adl`.
- Compile the .cpp file into a shared object library (.so) or dynamic link library (.dll).
- Copy the compiled file into the server's bin directory, and the .adl file into the server and Studio plugins directories.

The RSS adapter is an in-process adapter. For detailed instructions about compiling and linking an in-process adapter, see *Sybase CEP C/C++ SDK* on page 101.

SNMP Get OIDs Adapter

The Simple Network Management Protocol (SNMP) is an Internet-standard protocol for managing devices on IP networks. The SNMP Get OIDs Adapter enables you to receive SNMP information about system devices, network devices, and other devices.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description	Required?
SNMP version	Version	String	Indicates whether to use SNMP version 1, 2c, or 3. The Sybase CEP SNMP Get and Set adapters support only SNMP versions 1 and 2c.	Optional
Community String	Community-String	String	The SNMP password. The default value is "public". The SNMP community string will be transmitted unencrypted.	Optional

Property Name (screen)	Property Name (Attach Adapter)	Type	Description	Required?
Host machine	Host	String	Either an IP address or a symbolic address of the host machine. Sybase CEP does not address issues of firewalls in accessing the host machine. If the host machine is not supplied, it defaults to localhost.	Optional
OID list	OIDList	String	The Object ID (OID) of the device from which messages will be received. There may be an arbitrary number of OIDs entered into the OID list. It is your responsibility to ensure the SNMP datatype associated with the OID matches the Sybase CEP schema datatype.	Required
Seconds Per Poll	SecondsPerPoll	Integer	The specified OID will be polled with this period (in seconds). The five second default suffices for testing purposes, but a poll of a minute or more is probably adequate for real-life situations.	Optional

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by

the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

These properties determine which SNMP object(s) send or receive the data. The schema of the stream depends on the device you are getting information for. To determine the fields needed in the stream schema, you may need to look at the Management Information Base (MIB) that describes the device.

This Sybase CEP demonstration adapter uses the Net-SNMP `get_node` () call to poll the OID. Other calls such as `read_oid()` or `read_objio()` may be more appropriate to a specific application. `Get_node()` was chosen because it provides access to a wide variety of OIDs.

Sybase CEP Datatypes vs. SNMP Datatypes

SNMP supports data types that do not directly correspond to Sybase CEP data types.

For example, Sybase CEP does not have unsigned data types. Sybase CEP also does not support the SNMP Sequence and Sequence-of datatypes without user assistance.

Sybase CEP maps the SNMP datatypes as follows:

ASN.1 Datatype	Sybase CEP Datatype
ASN_INTEGER	C8_INT
ASN_OCTET_STR	C8_CHAR_PTR
ASN_BIT_STR	C8_BLOB_PTR
ASN_OPAQUE	C8_BLOB_PTR
ASN_OBJECT_ID	C8_BLOB_PTR
ASN_TIMETICKS	C8_INTERVAL
ASN_GAUGE	C8_INT
ASN_COUNTER	C8_INT
ASN_IPADDRESS	C8_INT
ASN_NULL	C8_BLOB_PTR
ASN_UINTEGER	C8_INT
ASN_COUNTER32	C8_LONG
ASN_COUNTER64	C8_LONG
ASN_OPAQUE_I64	C8_BLOB_PTR
ASN_OPAQUE_U64	C8_BLOB_PTR
ASN_OPAQUE_COUNTER64	C8_BLOB_PTR

ASN.1 Datatype	Sybase CEP Datatype
ASN_OPAQUE_FLOAT	C8_FLOAT
ASN_OPAQUE_DOUBLE	C8_FLOAT

When using SNMP, provide a fixed schema that takes the entire SNMP request and sends a single message to the Sybase CEP Engine. This single message will provide a time basis for the SNMP query.

Take the following into consideration:

- Columns that hold SNMP type Counter32 should be defined as Long rather than Integer to allow proper extraction. Otherwise, overflow can result in negative values.
- Sybase CEP has only signed integer and long values while ASN.1/net-snmp has both signed and unsigned values.
- Sybase CEP uses a C/C++ double for C8_FLOAT while ASN.1/net-snmp has both C/C++ float and double.
- ASN_TIMETICKS uses a resolution of hundredths of a second while Sybase CEP uses microseconds. This adapter multiplies the value by 10,000 to convert to microseconds. ASN_TIMETICKS is an interval, not a wall clock time, so the conversion to C8_INTERVAL is correct.
- The conversions to C8_BLOB_PTR are left to developers to decode as necessary. The coded interpretation is generally OID specific.

The object IDs may be in either of two forms: symbolic or numeric. Numeric OIDs contain embedded period characters (".") as part of the OID. If more than one OID is specified, the OIDs must be separated by a blank, tab, or newline.

SNMP Set Adapter

The SNMP Set Adapter enables you to send SNMP information. The SNMP is an Internet-standard protocol for managing devices on IP networks.

When the Sybase CEP Engine sends a message to the Set adapter, the only column present is a string.

A Protocol Data Unit is created and the adapter sends the contents of the message to the target OID.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description	Required?
SNMP version	Version	String	Indicates whether to use SNMP version 1, 2c, or 3. The Sybase CEP SNMP Get and Set adapters support only SNMP versions 1 and 2c.	Optional
Community String	Community-String	String	The SNMP password. The SNMP community string will be transmitted unencrypted.	Optional
SNMP Host machine	Host	String	Either an IP address or a symbolic address of the host machine. Sybase CEP does not solve issues of firewalls in accessing the host machine.	Optional
OIDs	OIDList	String	The OID to which messages are sent.	Optional

These properties determine which SNMP objects send or receive the data. The schema of the stream depends on the device that you are getting information for. To determine the fields needed in the stream schema, you may need to look at the Management Information Base (MIB) that describes the device.

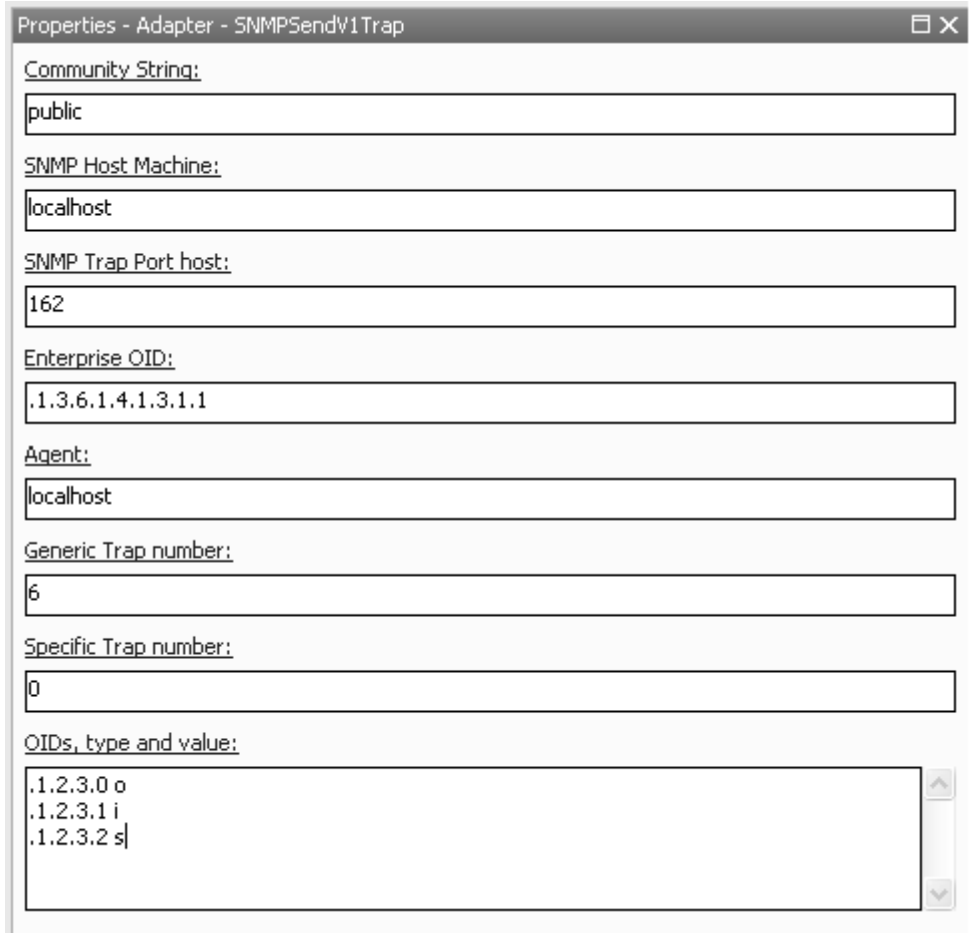
The OIDs can be in either of two forms: symbolic or numeric. Numeric OIDs contain embedded period characters (".") as part of the OID. If more than one OID is specified, the OIDs must be separated by a blank, tab, or newline.

For information about mapping Sybase CEP data types to SNMP data types, please see *Sybase CEP Datatypes vs. SNMP Datatypes* on page 494.

SNMP Send V1 Traps Adapter

The Sybase CEP Engine supports sending V1 traps.

The adapter's Properties View for V1 traps appears as:



The properties derive from the Net-snmp sendtrap utility. The syntax for this command corresponds closely to the adapter Properties View settings. The uptime in the sendtrap utility is determined from internal values and represents an approximation to the uptime of the system hosting Sybase CEP Server. The Generic Trap Number corresponds to the commonly used enterprise trap.

The variable length list in the "OIDs, Type and Value" field contains data sent with the trap. This list must conform to the schema definition. Since there are three OID/types in this list, there should be three data items in the schema. The actual data is converted to a string and appended to the OID list item. There is an assumption that the schema types maps into the Net-snmp types. This mapping is:

Abbreviation	Datatype
i	integer

u	unsigned32
c	counter32
t	Time ticks 1/100 second
a	IP address
o	OID
s, x, d	Octet string
n	Null
b	bit string
U	ASN_OpaqueU64
I	ASN_OpaqueI64
F	ASN_Opaque_Float
D	ASN_Opaque_Double

The datatype to be used depends on the datatype on the host machine of the OID. If numeric datatypes are used, it is not necessary to provide a valid MIB OID; a made-up OID as illustrated in the screenshot can suffice. A made-up OID places the burden of processing on the trap daemon on the host machine. This can be convenient as this corresponds closely to a "name value" pair on the host machine.

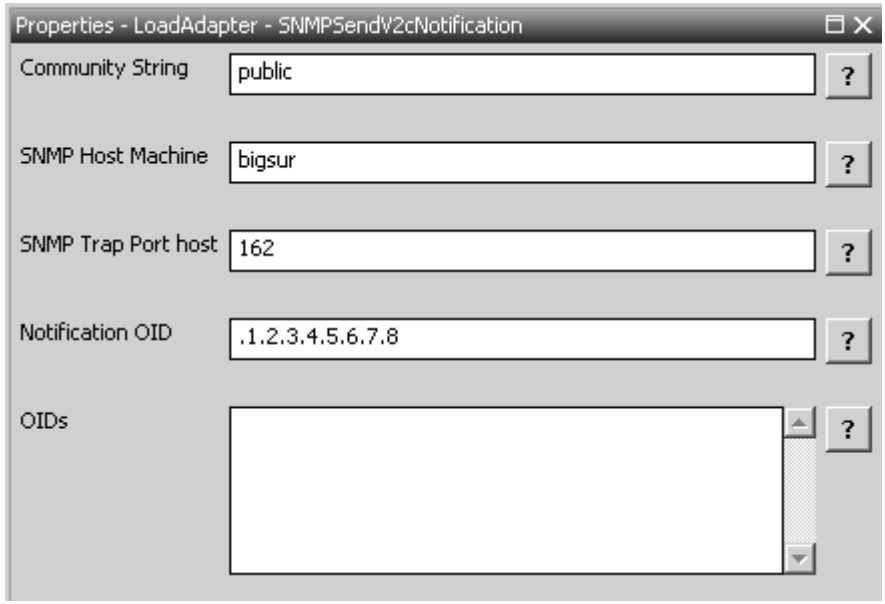
Using a symbolic OID requires the proper correspondence of datatypes.

The Sybase CEP schema should closely approximate the Net-snmp datatype. Since all data is first converted to a string before calling the Net-snmp libraries, this allows flexibility in datatype matching.

SNMP Send V2c Notifications

The Sybase CEP Engine supports sending V2c notifications.

The adapter Properties View for V2c notifications looks like the following:



The notification OID lets the SNMP host machine know the particular type of trap. The datatype abbreviations correspond exactly to the V1 datatypes, as does the Sybase CEP matching of schema values to Net-snmp values. See the table that describes *Net-SNMP trap datatypes* on page 496 for more information.

XML: Read From XML File

The Read From XML File adapter reads Rows from an XML file.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Filename	Filename	String	The name and path of the XML file to read data from. You can specify a file name and path either by typing it in or by clicking on the Browse button for this field. By default, the path is relative to the server's adapters base folder and must be underneath that base folder, but you can use $ProjectFolder to specify a path relative to the current project folder. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. For details about the Browse and Edit buttons to the right of the filename, see the discussion following this table.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Loop Count	LoopCount	Integer Min: 0 Max: 2000000000 Default: 1	If the Loop Count is 1, the file is read only once and the adapter stops sending data once the end of file is reached. If the Loop Count is greater than 1, then the data will be sent the specified number of times after the adapter finishes reading the file, the adapter starts reading again from the beginning. If the Loop Count is 0, the adapter repeats indefinitely.
Rate	Rate	Float Min: 0.0000001 (if set to a number greater than zero)	If this property is non-zero, then the adapter reads data from the input file at the specified rate (per second). Any row timestamps in the input file are ignored. The default is to leave this unset, in which case the row timestamp determines when each row is sent.
Timestamp Base	TimestampBase	Timestamp	The point at which time "starts" for this adapter, relative to the file's first timestamp. For example, if Timestamp base is 0 and the first row's timestamp is 5000000 (in microseconds), the first row is sent 5 seconds (5000000 microseconds) after the module starts. If blank, the first row is sent immediately after the module starts.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Set Timestamp To Current Time	UseCurrentTimestamp	Boolean	If set to true, the adapter overrides the row timestamp specified in the file with the current system time. Defaults to false.

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

A sample XML file with two rows is shown below:

```
<Tuple xmlns="http://www.sybase.com/cpx/2004/03/"
 Timestamp="946713600000001">
   <Field Name="ID">1</Field>
   <Field Name="BandName">Soft White Underbelly</Field>
</Tuple>
<Tuple xmlns="http://www.sybase.com/cpx/2004/03/"
 Timestamp="946713600000002">
   <Field Name="ID">2</Field>
   <Field Name="BandName">The Quarrymen</Field>
</Tuple>
```

The "http://www.sybase.com/cpx/2004/03/" part of the first line in each tuple should always be the same. Do not change the date.

The first tuple (row) is shown in lines 1-4, and the second tuple (row) is shown in lines 5-8.

The first line of each tuple contains the row timestamp, formatted as the number of microseconds since midnight January 1, 1970.

Lines two to three contain the field values for one row and lines 6-7 contain the field values for a separate row. Note also that the XML file is not necessarily formatted as shown in the example above because the usual format is more difficult to read.

Note: The values and column names are specified but the data types are not. The data must be appropriate for the data type of the column.

The ReadFromXMLFile adapter always has a 3-second startup delay before the first row is sent. If looping is used, the 3-second delay occurs only on the first iteration of the loop, not subsequent iterations.

Unless specified otherwise, the format of TIMESTAMP values, including the row timestamp, inside the file is a 64-bit signed integer representing the number of microseconds since the beginning of the epoch (midnight January 1, 1970 UTC/GMT). Unless you've specified otherwise for TIMESTAMP values, do not use formats such as "2007-01-09 12:30:34.567891

PDT" in the CSV file. For more information, see *Reading, Writing, and Converting Timestamps* on page 384.

The format of INTERVAL values inside the file may be any of the acceptable formats for INTERVAL values, including:

- A 64-bit signed integer representing a number of microseconds.
- [D [day[s]][ ][HH:MI[:SS[.FF]]] (for example, 1 02:03:04.000005).
- [D day[s]][ ][HH hour[s]][ ][MI minute[s]][ ][SS[.FF] second[s]] (for example, 1 day 2 hours 3 minutes 4.000005 seconds).

For a complete list of valid formats for INTERVAL values, see "INTERVAL Literals" in the Sybase CEP *CCL Reference Guide* .

Note: Although CCL statements require the keyword INTERVAL prior to the value for some of these forms, no INTERVAL keyword is required when using such values in CSV or XML files.

The Adapter Properties screen in Sybase CEP Studio allows you to specify an input path and file name for the file by clicking the Browse button and identifying the file. The path is relative to the adapters base folder, either as specified for Sybase CEP Server running on the same computer as Sybase CEP Studio, or as specified in the preferences for Sybase CEP Studio. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. In order for this feature to work properly, make sure that the Adapter's Base Folder field in Sybase CEP Studio Settings is set to the same folder as the adapters base folder for Sybase CEP Server. The Base Folder setting for the Sybase CEP Server is specified during the installation process, and can also be changed in the Sybase CEP Server's `c8-server.conf` file. The base folder setting for Sybase CEP Studio can be set from the Tools->Settings command on the Sybase CEP Studio menu.

The Edit button opens an editor that will allow you to edit any file whose name you enter into the filename field. This allows you to correct errors in the data. If the file's extension is "csv" or "xml", Sybase CEP Studio opens the appropriate editor specified in the "External Tools" tab available from the menu item "Tools -> Settings". For files with other extensions, on Microsoft Windows the editor is the one specified by the operating system's file associations. On UNIX-like operating systems, Sybase CEP Studio opens the editor specified by the EDITOR environment variable.

Note: When you click the Edit button, Sybase CEP Studio will look for the file in the SybaseC8Repository, even if the adapters base folder is set to another location. You may need to use the Browse button (adjacent to the Edit button) to navigate to the desired directory before you try to edit the file.

XML: Read from XML Socket

A *Read from XML Socket* adapter attempts to open a TCP connection to a given host and port address and, once connected, reads rows from the connection as XML.

A *Read from XML Socket* adapter attempts to open a TCP connection to a given host and port address and, once connected, reads rows from the connection as XML. If the connection is lost during execution, the adapter attempts to reconnect.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Host	Host	String	Host name or IP address of the data source.
Port	Port	Integer Min: 1 Max: 65535 Default: none	Port number of the data source.

XML: Write To XML File Adapter.

A Write To XML File adapter writes received rows to an XML file. If the file already exists, the adapter overwrites it.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Filename	Filename	String	The name and path of the XML file to write data to. You may specify a file name and path either by typing it in or by clicking on the Browse button for this field. By default, the path is relative to the server's Base Folder and must be underneath that base folder, but you can use $ProjectFolder to specify a path relative to the current project folder. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. For more information about the Browse and Edit buttons, see the discussion that follows this table.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Maximum Size in bytes	MaximumSize	Integer Min: 1	The maximum size of the output file. If this property is set, the adapter starts writing a new file every time the size of the current output file becomes greater than this property. The files are named <filename>, <filename>.001, <filename>.002, ... where <filename> is the value of the Filename property. If the property is not set, output is written to a single file that can grow as large as the available space.
Append To Existing File	Append	Boolean	This parameter affects how the adapter behaves when the adapter re-starts. If "Append To Existing File" is set to false, then when the adapter is started, the adapter moves any existing file named <Filename> to <Filename>.### and starts a new file named <Filename>. If a maximum size is specified and "Append To Existing File" is set to true, then when the adapter is started, the adapter appends to any existing file named <Filename> rather than immediately renaming the existing file.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Set timestamp to current time	UseCurrentTimestamp	Boolean	If set to true, the adapter overrides the timestamp specified in the row with the current system time. Defaults to false.
Formatted	Formatted	Boolean	If the flag is set to true, then the output XML will be indented and will include end-of-line characters. If the flag is set to false, then the output XML will not be indented.
Include Column Names	Extended	Boolean	If the flag is set to true, then field names will be included within each output message. If the flag is set to false, then field names will not be included within each output message.

For more information about the format of the data written to the XML file, see the description of the file format in the section about the "Read From XML File" adapter *XML: Read From XML File* on page 500. Note that when Sybase CEP Server writes INTERVAL values in XML format, the values are always written as a 64-bit signed integer representing a number of microseconds.

The Adapter Properties screen in Sybase CEP Studio allows you to specify an input path and file name for the file by clicking the Browse button and identifying the file. The path is relative to the adapters base folder, either as specified for Sybase CEP Server running on the same computer as Sybase CEP Studio, or as specified in the preferences for Sybase CEP Studio. For more information, see *Setting The Base Folder For File Input/Output Adapters* on page 377. In order for this feature to work properly, make sure that the Adapter's Base Folder field in Sybase CEP Studio Settings is set to the same folder as the adapters base folder for Sybase CEP Server. The Base Folder setting for the Sybase CEP Server is specified during the installation process, and can also be changed in the Sybase CEP Server's `c8-server.conf` file. The base folder setting for Sybase CEP Studio can be set from the Tools->Settings command on the Sybase CEP Studio menu.

The Edit button brings up an editor that will allow you to view any file whose name you enter into the filename field. If the file's extension is "csv" or "xml", Sybase CEP Studio opens the appropriate editor specified in the "External Tools" tab available from the menu item "Tools ->

Settings". For files with other extensions, on Microsoft Windows the editor is the one specified by the operating system's file associations, and on UNIX-like operating systems, Sybase CEP Studio opens the editor specified by the EDITOR environment variable.

Note: When you click the Edit button, Sybase CEP Studio looks for the file in the SybaseC8Repository, even if the adapters base folder is set to another location. You may need to use the Browse button (adjacent to the Edit button) to navigate to the desired directory before you try to open the file.

XML: Write to XML Socket

A *Write to XML Socket* adapter attempts to open a TCP connection to a given host and port address and, once connected, writes rows to the connection as XML.

If the connection is lost during execution, the adapter attempts to reconnect.

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
Host	Host	String	Host name or IP address of the data destination.
Port	Port	Integer Min: 1 Max: 65535 Default: none	Port number of the data destination.

XML: Write XML Over HTTP Adapter.

For each message this adapter receives, it posts the message in XML format to an HTTP server

Property Name (screen)	Property Name (Attach Adapter)	Type	Description
URL	URL	String	The URL to which the message should be posted.
XSL Template	XSLTemplate	String	XSL Template for converting XML message. This property is optional. See the description below.

The XSL template can be used for converting the raw XML message to some other format.

To convert the raw XML to a SOAP call, you can use an XSL template similar to the one below. This example is based on raw data from Sybase CEP that consists of two fields per message, Symbol and Price):

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:c8=http://www.sybase.com/cpx/2004/03/
xmlns:xsl=http://www.w3.org/1999/XSL/Transform version="1.0">
<!-- Specify lookup key -->
<xsl:key name="FieldName" match="c8:Tuple/c8:Field" use="@Name" />
<xsl:template match="/">
<soap:Envelope xmlns:soap=http://www.w3.org/2001/12/soap-envelope
       soap:encodingStyle=http://www.w3.org/2001/12/soap-encoding>
  <soap:Body xmlns:m=http://www.example.org/stock>
    <m:SetStockAlert>
      <m:StockName>
         <xsl:value-of select="key('FieldName','Symbol')"/>
      </m:StockName>
      <m:StockPrice>
         <xsl:value-of select="key('FieldName','Price')"/>
      </m:StockPrice>
    </m:SetStockAlert>
  </soap:Body>
</soap:Envelope>
</xsl:template>
</xsl:stylesheet>
```

If more customization is needed (for example, for custom HTTP headers), you can copy and modify the adapter code in the file `.../sybasec8/server/sdk/c/examples/c8_write_xml_over_http.cpp`

To create a customized version of this adapter, perform the following:

- Copy and modify the files `c8_rss_reader.cpp` and `c8_rss_reader.adl`.
- Compile the .cpp file into a shared object library (.so) or dynamic link library (.dll).
- Copy the compiled file into the server's bin directory. Copy the .adl file into the server and Studio plugins directories.

This is an in-process adapter. This code can be modified and compiled into an in-process adapter. For more information about how to compile in-process adapters written in C, see *Sybase CEP C/C++ SDK* on page 101.

Windows Event Log Adapter

The Windows Event Log adapter reads from a Microsoft Windows event log and publishes the event information to a Sybase CEP stream.

The adapter works with Microsoft Windows XP and Vista. The adapter supports the Report-Events style logging found in Windows XP. It does not support the additional features—such as user defined fields, Setup, and Forwarded events—that were added to the Microsoft Windows Event Log in Microsoft Windows Vista.

The Windows Event Log adapter must be placed on the machine that hosts the log from which you wish to read. The adapter publishes to a Sybase CEP stream. The computer running the

adapter runs can be either the same or different from the machine that Sybase CEP Server is running on. In other words, you can monitor events on a different computer than Sybase CEP Server is running on.

The adapter only reports events that occur while the adapter is active. The adapter does not publish the part of the event log that was written before the adapter started.

Installing the Windows Event Log adapter

The Windows Event Log adapter requires the Microsoft .NET 2.0 Framework.

The adapter is supported on Microsoft Windows XP and Vista.

If you want to use the adapter as a Microsoft Windows service, you must register the service using `installutil.exe` from the .NET Framework 2.0 by executing the command:

```
installutil WinEventLogAdapterService.exe
```

Note: To uninstall the adapter as a service, run the installutil program with the "/u" flag:**installutil /u WinEventLogAdapterService.exe**

If you do not want to use the adapter as a Microsoft Windows service, then you do not need to do any special installation steps.

Usage

This adapter is an out-of-process adapter. The adapter may be run as a stand-alone program or as a Microsoft Windows service.

The `WinEventLogAdapterService.exe` file is in the `server/sdk/net` subdirectory. If you install Sybase CEP Engine to the default directory, this will be `C:\Program Files\C8\Server\sdk\net`.

When the adapter is started, it takes the following arguments:

Description	Example
The URI of the Sybase CEP stream where the events are to be sent.	`ccl://localhost:6789/Stream/Wkspc/WinEvt/InEvents`
One or more event logs that the adapter will monitor.	`Security`

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

To start the adapter as a stand-alone program, execute a command similar to the following:

```
WinEventAdapter ccl://localhost:6789/Stream/Workspace/
   WinEvtAdapter/InEvents Security Application
```

Although this may be displayed as more than one line, this should all be on one line.

This command starts the adapter, tells the adapter to publish to the specified stream URI, and tells the adapter to read events from two Microsoft Windows event logs: Security and Application.

As a stand-alone program, the adapter will run continuously until another process sends the adapter the KILL signal.

To run the adapter as a windows service, go into the Service Explorer by clicking Start -> Settings -> Control Panel -> Administrative Tools -> Services. Start the Windows Event Adapter Service using the same arguments as for the stand-alone application. You can also edit the file `WinEventLogAdapterService.exe.config`, located in the same directory as the adapter executable, to provide configuration parameters. You can specify four parameters:

- StreamUri: (Required). The URI of the stream where the adapter will publish events.
- EventLog: (Required). The names of the event logs to monitor, separated by commas.
- RetryLimit: (Optional). How many times the adapter should retry publishing to a stream before giving up (used when the server stops and takes some time restarting). Defaults to 0, indicating no retries.
- RetryIntervalSeconds: (Optional). How long to wait between retry attempts. Defaults to 60 seconds.

Examine the sample file provided with the adapter for formatting specifics.

Schema

A description of the schema used for messages sent by the Windows Event Log adapter.

Field	Type	Description
Category	String	Text associated with the Category Number.
CategoryNumber	Integer	Category of the event log entry.
Data Blob	Binary	Data associated with the event.
EntryType	String	May be one of the following: • Error. • Warning. • Information. • SuccessAudit. • FailureAudit.

Index	Integer	Index of the entry in the event log.
InstanceID	Long	Resource identifier that identifies the message text of the event.
MachineName	String	Machine where the event occurred.
Message	String	Text message associated with the event.
Source	String	Application that generated the event.
TimeGenerated	Timestamp	Time that the event occurred (UTC, not local time).
TimeWritten	Timestamp	Time that the event was written to the log (UTC, not local time).
User_Name	String	Name of the user responsible for the event.
LogName	String	Event Log in which the event was generated.
Site	String	Text name of the site of the event.

The event schema may also be found in the **wineventlog.ccs** file. If you installed Sybase CEP Engine to the default directory, this file will be in the **C:\Program Files\C8\Server\sdk\net\ccl** directory.

The timestamp of the event is the time that the event was generated. The time that the event was written is recorded separately, within the message.

For further information, see the documentation for the EventLogEntry class on MSDN.

SNMP Adapter Information

This chapter provides an overview of the SNMP adapter.

Motivation for SNMP

The Simple Network Management Protocol (SNMP) is an Internet-standard protocol for managing devices on IP networks.

Many devices support SNMP and uses range from ordinary to exotic. Monitoring the health of these devices can provide useful functionality to a Sybase CEP system. Adding the SNMP capabilities to control network devices, page someone, or take automatic actions on pre-programmed events makes SNMP an important part of an administrator's toolkit.

Sybase CEP allows you to both get and set SNMP OIDs.

Configuring Your Environment for SNMP Adapters

The Sybase CEP distribution assumes that you have an existing SNMP implementation, including a properly configured and working daemon or service. The configuration files for Net-snmp 5.4 are included in the Sybase CEP distribution.

In order to preserve existing SNMP configuration files, environment variables are used to append the Sybase CEP Server information to the user 's configuration. Care has been taken to not overwrite user configuration files but, as in all similar installation processes, you should back up your files before the files are modified. Please refer to *http://net-snmp.sourceforge.net/docs/man/snmp_config.html* and similar installation documentation for a thorough discussion of Net-snmp configuration.

The environment settings for Sybase CEP Server are provided in source code form in the file `c8_snmp_common.c` which can be modified as needed. This initial directory configuration may not be adequate for most needs. In particular, the configuration does not provide for community string settings, authorizations, MIBs for the large variety of hardware available, and so on. These environment variables have been exposed in this manner in an attempt to accomodate the various build-time configurations available to Net-snmp 5.4. The variables documented below are hard-coded into the original source Net-snmp 5.4 by the configuration process at build time. By patching the Net-snmp 4.5 source code with these environment variables, you can create your own configurations.

An effort has been made to ensure proper Net-SNMP operations for a wide variety of configurations. The reason Sybase CEP "soft-coded" these variable is that the net-snmp installation hard-codes these variables at compile time. You may have different settings and

the enviromental override scheme is an attempt to accomodate these differences. You should not have to adjust your Net-SNMP settings for Sybase CEP Server.

Because the environmental settings are within Sybase CEP Server process, these settings do not affect other processes.

The Sybase CEP implementation of SNMP assumes a daemon is running. The Net-SNMP service cannot co-exist with the Windows SNMP service unless specific configuration instructions are followed. Refer to the `README.win32` instructions for this installation configuration. Your are responsible for this and all other Net-SNMP installation configurations.

In the following list, the C8_INSTALL is the directory in which Sybase CEP Server is installed. The default locations of Net-SNMP files and directories are assumed.

- `INSTALL_BASE=C8_INSTALL/snmp`

  This is the snmp directory in the installation directory of Sybase CEP Server. This value is normally set to `C:/usr` for Windows and `/usr/local` for Linux; the same defaults in the source distribution. By setting this value to C8_INSTALL/snmp, you insure that your configuration and MIB files cannot be overwritten.
- `NETSNMP_DEFAULT_MIBS=C:/usr/share/snmp/mibs; D:/usr/share/snmp/mibs; C8_INSTALL/mibs`

  This is the location of the MIBs directory. The Net-snmp defaults are first. This is because you are expected to maintain your own MIBs. The Sybase CEP MIBs cannot cover all possible hardware configurations. On Linux the default MIB directory is `/usr/local/share/snmp/mibs` . MIBs directories, just as in Net-snmp, must contain MIBs and only MIBs. Non-MIB files are likely to cause syntax errors that will crash the MIB parser.
- `NETSNMP_PERSISTENT_DIRECTORY=C8_INSTALL/snmp`

  This directory is used only by the snmpd daemon and is not expected to be used in the Sybase CEP Server environment. This corresponds to the SNMP_PERSISTENT_DIR variable.
- SNMPLIBPATH=C8_INSTALL/snmp

  This is the location of the net-snmp libraries. Overriding this variable should be performed with caution if your version of net-snmp is not Net-snmp 5.4.
- `SNMPSHAREPATH=C8_INSTALL/snmp`

  The shared path. On linux this is `/usr/local/share/snmp`
- `SNMPCONFPATH=C8_INSTALL/snmp`

  A list of directories to search for configuration files. Refer to *http://net-snmp.sourceforge.net/docs/man/snmp_config.htm* for more information.
- `SNMPDLMODPATH=C8_INSTALL/snmp`.

  The dynamically linked library path. See *http://net-snmp.sourceforge.net/docs/man/snmp_config.html* for more information.
- `NETSNMP_TEMP_FILE_PATTERN=C8_INSTALL/snmp/snmpdXXXXXX`

The temp file pattern for the `mktemp()` function.

Configuring the c8-server.conf File for SNMP

SNMP adapters usually need to reference information in Management Information Bases (MIBs). You can set a preference in the `c8-server.conf` file to specify where those MIBs should be read from.

Net-snmp uses the "MIBDIRS" environment variable to specify which directories to read when loading MIBs at startup. You can set a value to tell the server to add directories to that MIBDIRS environment variable.

The section of `c8-server.conf` relevant to these preferences is in the "SybaseC8/Adapters" section and looks similar to:

```
<section name="SNMP">
    <!-- The semi-colon separated list of folders with MIBS for SNMP
-->
    <preference name="MIBDIRS" value="C:\Program Files
\SybaseC8\Server\bin\mibs"></preference>
</section>
```

The preference "SNMP"/"MIBDIRS" allows you to load additional MIBs; if the MIBDIRS preference is set, then its value will be concatenated to the semicolon-separated list of directories of the existing "MIBDIRS" environment variable. (If the "MIBDIRS" environment variable does not exist, it will be created.)

If this preferences section does not exist, then by default the server will load the MIBs in the directories specified in the MIBDIRS environment variable.

Sybase CEP Drivers

Sybase CEP provides a number of custom drivers, such as database drivers. Note that many major database vendors, such as Oracle and MySQL, supply their own drivers, so Sybase CEP does not supply drivers for them.

To use a Sybase CEP driver to communicate with an outside entity, such as a database server, you must configure the `c8-services.xml` file. Please see the Sybase CEP *Installation Guide* for instructions.

Status Information

An appendix that provides information about Message Groups and Message Names.

Message Groups and Message Names were previously described in the section titled *Monitoring Servers and Projects* on page 34.

Status Information about a CCL Application

Information on interpreting status information about a CCL Application.

Description: CclApplicationInfo status messages contain runtime information about a given CCL Application ("Project").

ObjectID:The ObjectID is the full path to the given CCL in the form:

```
<WorkspaceName>/<ApplicationName>
```

where ApplicationName is the top module's load name.

Name of MessageGroup:CclApplicationInfo

MessageName	Value Type	Message Availability and Frequency	Description
AutoRestartAttempt	Long	Sent when an automatic module restart occurs (auto-restart must be configured).	The number of times the module has been auto-restarted. The number is reset back to zero if the user manually restarts module or if the container is restarted.
DroppedMessages	Long	By default, once per second.	The total number of Dropped out-of-order and late messages for the project
InputMessages	Long	Everywhere, updated approximately every 1 second.	How many messages the application received. Sum of output messages for all external input streams.

IsLoadSavedState	Boolean	Status object only, never updated.	False if module was started 'from clean slate' True otherwise
LastError	String	Everywhere, updated on event	Event: application gets an error.
LoadedAt	Timestamp	Status object only, never updated.	When the application was loaded into the Manager.
ManagerBuildDate	String	Status object only, never updated.	The Manager's build date.
ManagerURI	String	Status object only, never updated.	The Manager's URI.
ManagerVersion	String	Status object only, never updated.	The Manager's version string.
OutputMessages	Long	Everywhere, updated approximately every 1 second.	How many messages the application sent. Sum of input messages for all external output streams.
OwnerName	String	Status object only, never updated.	The UserName of the User who started this particular project. This message is only available for projects that are running, and only if user authentication is enabled.
PendingMessages	Long	Everywhere, updated approximately every 1 second.	How many messages the application needs to process. This is a sum of pending messages number for all CCX modules in the corresponding program.

PendingPersistentMessages	Long	Everywhere, updated approximately every 1 second.	How many messages the application needs to persist on disk. This is a sum of pending messages number for all CCX modules in the corresponding program.
ProjectLatency	Long	Everywhere, updated approximately 1 time per second.	The average latency of the CCL query, excluding adapters and other queries. See also "SystemLatency".
RequestedState	String	Everywhere, updated approximately every 1 second.	For how long the application is running (available only if application is running).
RpcFailedMessages	Long	Updated approximately every 1 second.	The total number of failed RPC calls for the project.
RpcSentMessages	Long	Updated approximately every 1 second.	The total number of successful RPC calls for the project.
RunningTime	Interval	Status object only, never updated.	For how long the application is running (available only if application is running).
StartedAt	Timestamp	Status object only, never updated.	When the application was started (available only if application is running).
State	String	Everywhere, updated on event.	Event: application changes it state: - Unloaded - Unregistered - Unregistering - Registered - Registering - RegisterFailed - Started - Starting - StartFailed

SystemLatency	Long	Everywhere, updated approximately 1 time per second.	The average latency of the CCL query including adapters and other queries. See also "ProjectLatency".
VersionString	String	Status object only, never updated.	The application's version string generated from CCL sources.

Status Information about CCL Compiler Settings

CclCompilerInfo status messages contain information about Sybase CEP Compiler options, and are used when the given CCL Application is compiled into a CCX Program.

ObjectID: The ObjectID is the full path to the given CCL Application in the form:

`<WorkspaceName>/<ApplicationName>` Where `ApplicationName` is the top module's load name.

Name of MessageGroup: CclCompilerInfo.

MessageName	Value Type	Message Availability and Frequency	Description
CclFile	String	Status object only, never updated.	The path to the top module's CCL file relative to SybaseC8Repository, specified by RepositoryPath.
CcxFile	String	Status object only, never updated.	The path to the CCX file relative to SybaseC8Repository, specified by RepositoryPath.
CclName	String	Status object only, never updated.	The load name, or application name, for the top module.
Other Compiler command line parameters (see compiler help for details).	String	Status object only, never updated.	The compiler parameters used.

MessageName	Value Type	Message Availability and Frequency	Description
RepositoryPath	String	Status object only, never updated.	The path to the SybaseC8Repository. The repository path name is the root for relative pathnames, for example, for the .ccp and .ccl files.

Status Information for a CCL Query

CclQueryInfo status messages contain runtime information about a given CCL Query.

ObjectID: The ObjectID is the full path to the given Query in the form:

`<WorkspaceName>/<FullCclPath>/<StatementNumber>`

Where <FullCclPath> and <StatementNumber> are assigned to the query by the compiler.

Name of MessageGroup: CclQueryInfo

MessageName	Value Type	Message Availability and Frequency	Description
InputMessages	Long	Everywhere, updated approximately every 1 second.	How many messages the query received (sum of input messages count for all query's "input" ccx primitives).
RpcFailedMessages	Long	Everywhere, updated approximately every 1 second.	The total number of failed RPC calls for the given CCL query.
RpcSentMessages	Long	Everywhere, updated approximately every 1 second.	The total number of successful RPC calls for the given CCL query.

OutputMessages	Long	Everywhere, updated approximately every 1 second.	How many messages the query sent (sum of output messages count for all query's "output" ccx primitives).

Status Information about CCL Stream Pairs

CclStreamPairInfo status messages contain information about Sybase CEP streams.

ObjectID: The ObjectID is the full path to the first stream in the pair and is provided in the form:

```
ccl://<hostname>:<portNumber>/Stream/<WorkspaceName>/
<ApplicationName> /<StreamName>
```

Where `ApplicationName` is the project's (in other words, the top module's) load name.

ObjectID2: The ObjectID2 is the full path to the second stream in the pair and is provided in the form:

```
ccl://<hostname>:<portNumber>/Stream/<WorkspaceName>/
<ApplicationName> /<StreamName>
```

Where `ApplicationName` is the project's (in other words, the top module's) load name.

Name of MessageGroup: CclStreamPairInfo

MessageName	Value Type	Message Availability and Frequency	Description
Latency	Long	The message is updated approximately every 1 second.	The value is the average latency between ObjectID1(stream1) and ObjectID2 (stream2).

Status Information about a Workspace

CclWorkspaceInfo status messages contain information about a given Workspace.

ObjectID:

```
<WorkspaceName>
```

Name of MessageGroup: CclWorkspaceInfo.

MessageName	Value Type	Message Availability and Frequency	Description
Description	String	Status object only, never updated.	The description that was specified for the workspace when it was created.
ManagerBuildDate	String	Status object only, never updated.	The Manager's build date.
ManagerURI	String	Status object only, never updated.	The Manager's URI.
ManagerVersion	String	Status object only, never updated.	The Manager's version string.

Status Information about a Container

ContainerInfo status messages contain runtime information about the specified Container.

The objectID is the Container's URI:

```
http://hostname:port/Container
```

or

```
https://hostname:port/Container
```

Name of MessageGroup: ContainerInfo.

MessageName	Value Type	Message Availability and Frequency	Description
BuildDate	String	Status object only, never updated.	The Container's build date.

MessageName	Value Type	Message Availability and Frequency	Description
ContainerAdded	String	Status Stream Only, updated on event.	Event: Container added (value =

MessageName	Value Type	Message Availability and Frequency	Description
			"Active" or "Passive").

MessageName	Value Type	Message Availability and Frequency	Description
ContainerKilled	String	Status Stream Only, updated on event.	Event: Killing container (value

MessageName	Value Type	Message Availability and Frequency	Description = reason).

MessageName	Value Type	Message Availability and Frequency	Description
ContainerRemoved	String	Status Stream Only, updated on event.	Event: Container removed (value

MessageName	Value Type	Message Availability and Frequency	Description = reason).

MessageName	Value Type	Message Availability and Frequency	Description
CPUTime	Interval	Everywhere, updated approximately every 1 second.	The amount of CPU Time used by t

MessageName	Value Type	Message Availability and Frequency	Description
			he Sybase CEP Server process sin

MessageName	Value Type	Message Availability and Frequency	Description
			ce it started.

MessageName	Value Type	Message Availability and Frequency	Description
CPUUtilization	Float	Everywhere, updated approximately every 1 second.	The percentage of CPU utilized b

MessageName	Value Type	Message Availability and Frequency	Description
			y t h e S y b a s e C E P S e r v e r p r o c e s s

MessageName	Value Type	Message Availability and Frequency	Description (1 = 100%).

MessageName	Value Type	Message Availability and Frequency	Description
HeartbeatPeriod	Interval	Status object only, never updated.	How often the container sends the

MessageName	Value Type	Message Availability and Frequency	Description
			artbeats.

MessageName	Value Type	Message Availability and Frequency	Description
LoadLimit	Long	Status object only, never updated.	The container load limit, or the

MessageName	Value Type	Message Availability and Frequency	Description
			max number of CCX modules allow

MessageName	Value Type	Message Availability and Frequency	Description
			ed to run.

MessageName	Value Type	Message Availability and Frequency	Description
LogErrorCount	Long	Everywhere, updated approximately every 1 second.	The total number of errors that

MessageName	Value Type	Message Availability and Frequency	Description
			have been logged in the server l

MessageName	Value Type	Message Availability and Frequency	Description
			og for a given container. This n

MessageName	Value Type	Message Availability and Frequency	Description
			umber is reset to 0 if the conta

MessageName	Value Type	Message Availability and Frequency	Description
			inner is restarted.

MessageName	Value Type	Message Availability and Frequency	Description
LogWarningCount	Long	Everywhere, updated approximately every 1 second.	The total number of warnings tha

MessageName	Value Type	Message Availability and Frequency	Description
			t h a v e b e e n l o g g e d i n t h e s e r v e r

MessageName	Value Type	Message Availability and Frequency	Description
			log for a given container. This

MessageName	Value Type	Message Availability and Frequency	Description
			number is reset to 0 if the con

MessageName	Value Type	Message Availability and Frequency	Description
			tainer is restarted.

MessageName	Value Type	Message Availability and Frequency	Description
ManagerURI	String	Status object only, never updated.	The Manager's URI this Container

MessageName	Value Type	Message Availability and Frequency	Description
			is connected to.

MessageName	Value Type	Message Availability and Frequency	Description
StartTime	Timestamp	Status object only, never updated.	The container's start time.

MessageName	Value Type	Message Availability and Frequency	Description
State	String	Status object only, never updated.	Container state (Active or Passi

MessageName	Value Type	Message Availability and Frequency	Description
			ve). Note: Only available through

MessageName	Value Type	Message Availability and Frequency	Description
			h m a n a g e r ' s G e t C o n t a i n e r S t a t u s a

MessageName	Value Type	Message Availability and Frequency	Description
			nd GetManagerStatus calls.

MessageName	Value Type	Message Availability and Frequency	Description
TotalMemory	Long (bytes)	Everywhere, updated approximately every 1 second.	The total memory available to Sy

MessageName	Value Type	Message Availability and Frequency	Description
			base CEP Server, in bytes.

MessageName	Value Type	Message Availability and Frequency	Description
UsedMemory	Long (bytes)	Everywhere, updated approximately every 1 second.	The total memory used by Sybase

MessageName	Value Type	Message Availability and Frequency	Description
			CEP Server, in bytes.

MessageName	Value Type	Message Availability and Frequency	Description
Version	String	Status object only, never updated.	The Container's version string.

Status Information about a Manager

ManagerInfo status messages contain runtime information about a given Manager.

ObjectID: The objectID is the Manager's URI:

```
http://hostname:port
```

or

```
https://hostname:port
```

Name of MessageGroup: ManagerInfo.

MessageName	Value Type	Message Availability and Frequency	Description
BuildDate	String	Status object only, never updated.	The Manager's build date.
Version	String	Status object only, never updated.	The Manager's version string.
StartTime	Timestamp	Status object only, never updated.	When the Manager was started.
ManagerHA PromotedToPrimary	String	Manager status stream only, updated on event.	Event: manager HA node promoted to primary (value = reason).
ManagerHA DemotedToBackup	String	Manager status stream only, updated on event.	Event: manager HA node demoted to backup (value = reason).
ManagerHA ParticipatingInElection	String	Manager status stream only, updated on event.	Event: manager HA participating in primary elections (value = reason).

Daylight Saving Time and the Sybase CEP Time Zone Database

This appendix describes how to configure the Sybase CEP Engine to deal with time zone issues.

Topics include:

- Changing the start and end dates of Daylight Savings Time.
- Creating custom time zones.
- Customizing information, such as time zone abbreviations, for existing time zones.

Time Zones

A time zone is a region of earth that has adopted the same standard time, usually referred to as the local time.

Most time zones are exactly one hour apart. By convention, all time zones compute their local time as an offset from GMT/UTC. GMT (Greenwich Mean Time) is an historical term originally referring to mean solar time at the Royal Greenwich Observatory in Britain. GMT has been replaced by UTC (Coordinated Universal Time), which is based on atomic clocks. Due to political and geographical practicalities, time zone characteristics may change over time. For example, the start date and end date of daylight savings time may change, or new time zones may be introduced to handle the needs of newly created countries. For all purposes, GMT and UTC are equivalent. For details about GMT and UTC, see the U.S. Naval Observatory web page: *http://www.usno.navy.mil/USNO/astronomical-applications/astronomical-information-center/world-tzones*.

Internally, Sybase CEP always stores timestamp information as a number of microseconds since midnight January 1, 1970 GMT. When data is converted from internal format to external format (for example, to a string in a form such as "YYYY-MM-DD HH24:MI:SS.FF"), Sybase CEP takes into account the time zone and generates the correct local time (according to the clock and time zone information stored in the computer). Similarly, when converting from a string to the internal format, Sybase CEP also takes into account the local time zone and GMT. To allow you to specify information such as whether their local time zone uses daylight savings time, Sybase CEP provides a time zone "database" that customers can customize. You may need to change this database under certain circumstances, including:

- Changes in the start and end dates of Daylight Savings Time.
- Addition of time zones that are not included in the default list provided by Sybase CEP.
- A desire to use different abbreviations (for example, "PDT") for certain time zones, to avoid duplicate abbreviations.

Other customizations are also possible.

Sybase CEP's time zone database is in the form of a CSV file named `c8_timezones.csv`, which contains one row per time zone. The Sybase CEP Server and Sybase CEP Studio each have a copy of this file in their respective plugins directories. For example, on Microsoft Windows:

```
C:\Program Files\SybaseC8\Server\plugins\sybasec8_timezones.csv
C:\Program Files\SybaseC8\Studio\plugins\sybasec8_timezones.csv
```

and on UNIX-like operating systems:

```
/home/<userid>/sybasec8/server/plugins/sybasec8_timezones.csv
/home/<userid>/sybasec8/studio/plugins/sybasec8_timezones.csv
```

The reason that the file is located in both the Sybase CEP Studio and server plugins directories is that the server and studio may be running on separate computers, and both the server and studio must interpret and display time zones. Obviously, if one time zone database file is changed, the changes should be copied to the other plugins directory(s).

Default Time Zones

Unless time zones are specified as part of timestamp information, both input and output timestamps are in local time.

Users can specify timezone information in various CCL function calls (for example, TO_TIMESTAMP()) and in some Sybase CEP SDK calls. For example:

```
... TO_TIMESTAMP("2002-06-18 13:52:00.123456 PST", "YYYY-MM-
DD HH24:MI:SS.ff TZD") ...
```

In this example, "TZD" tells the function to look for a time zone designator such as the "PST" at the end of the first string. The "PST" indicates that the time zone is Pacific Standard Time, assuming that you have not changed the default abbreviations in the Sybase CEP time zone database.

You must run Sybase CEP software on machines with proper time zone configurations.

Daylight Savings Time

If you specify a particular time zone that uses daylight savings time, Sybase CEP software takes this into account. Specifically, Sybase CEP looks at the time zone database file, reads the starting and ending dates for daylight savings time, and takes those dates into account when converting times from external format to the internal timestamp format. For example, the following 2 lines produce the SAME internal timestamp, even though one line specifies PST and the other specifies PDT:

```
TO_TIMESTAMP("2002-06-18 13:52:00.123456 PST",
   "YYYY-MM-DD HH24:MI:SS.ff TZD")
TO_TIMESTAMP("2002-06-18 13:52:00.123456 PDT",
   "YYYY-MM-DD HH24:MI:SS.ff TZD")
```

If a time zone designator is not used, then local time is applied. Again, daylight savings time will is taken into account if the local time zone uses daylight savings time and the specified timestamp is in the time period covered by daylight savings time.

If the start and end dates for daylight savings time change, you must change the Sybase CEP Time Zone Database to take into account the new start and end dates.

Note: The United States changed its start and end dates for daylight savings time; the change took effect in 2007. If you are using Sybase CEP Engine 4.6.2 or earlier and use U.S. time zones in any data, you should check and, if necessary, update the Sybase CEP Time Zone Database to take into account the new dates. Information about changing the database is included later in this appendix.

Transitioning from Standard Time to Daylight Savings Time and Vice-Versa

During the transition to/from daylight savings time, certain times do not exist. For example, in the U.S., during the spring transition from standard time to daylight savings time, the wall clock jumps from 01:59 to 03:00 and the time of 02:00 does not exist. Conversely, in the fall, 01:00 to 01:59 appears twice in one night because time jumps backward from 2:00 to 1:00 when daylight savings time ends.

However, because these undefined and/or ambiguous times can still be input, the Sybase CEP software has been developed to deal with them in some manner. During the transition to daylight savings time, the time from 02:00 to 02:59 does not exist as the wall clock jumps from 01:59 to 03:00. Sybase CEP will interpret 02:59 PST as 01:59 PST. In the fall, when daylight savings time disappears, ambiguous times exist because at 02:00, time jumps backwards one hour. Thus, Sybase CEP will interpret 02:00 PDT as 01:00 PST on this morning.

If you are truly concerned with these matters use GMT times, which do not include daylight savings time.

Duplicate Time Zone Abbreviations

There are many more than 24 time zones on earth. Some countries, such as India, have time zones that are not a multiple of an hour different from GMT. The Sybase CEP Time Zone database contains approximately 40 commonly-used time zones, and this is not a complete list.

In part because there are so many time zones, two or more time zones may have the same abbreviation. For example, both Australia and the U.S. use Eastern Standard Time (EST) to refer to the time zone on the eastern side of their respective countries. Because each time zone abbreviation in the Sybase CEP time zone database must be unique, Sybase CEP users in the U.S. may prefer to keep EST as a reference to U.S. Eastern Standard Time and use a different abbreviation (for example, "AEST") for Australia's Eastern Standard Time. Users in Australia, of course, may prefer to do the converse. Since Sybase CEP's time zone database is

user-customizable, you can choose whatever abbreviations you wish, as long as each time zone's abbreviations do not overlap the abbreviations for another time zone.

Sybase CEP Time Zone Database

Sybase CEP stores information about each time zone, including the abbreviation for the time zone, the start date of Daylight Savings Time and the end date of Daylight Savings Time, in a table of Comma-Separated Values.

We loosely refer to this table, and the file that contains it, as the Sybase CEP time zone database. The file is named `c8_timezones.csv` and a copy is stored in both the server plugins directory and the studio plugins directory.

There are eleven fields in the Sybase CEP Sybase CEP time zone database. The fields are separated by commas. The first line of the time zone database must be a title line containing the column names. This line must be present and contain eleven fields. This title line serves as a memory jog for the fields in the time zone database.

Each line is expected to have eleven fields. Some of these fields can be empty. While fields in the default file are quoted with double quotes, this is not mandatory. An empty field is designated by two successive commas, or by using double quotes, as shown in the 2 examples below:

```
,,
,"",
```

Empty lines are ignored. Lines beginning with '#' are comments and are also ignored. Users are encouraged to supply comment and spacer lines.

Some of the 11 fields are lengths of time. Some are date rules that they indicate when daylight savings time starts. Before we describe each of the 11 fields in each row of the database, we will describe the format for lengths of time, and the format for date rules.

Lengths of Time

Some fields represent a length of time.

The format of these fields must be: {+|-}hh:mm[:ss] where the leading sign is optional, "hh" represents a number of hours, "mm" represents a number of minutes, and "ss" represents a number of seconds. The "ss" for seconds is optional. If the value is negative, the leading "-" is, of course, mandatory. Leading zeroes for hours and minutes are optional.

Date Rules

A data rule is a specially formatted string used to indicate a day of the year for transitions, for example, transitions to or from Daylight Savings Time.

A date rule indicates dates such as "the second Sunday in April" or "the last Sunday in October". These are typical descriptions of daylight savings time transitions. Date rules consist of three subfields:

- "nth" weekday of the month. For example, the 3rd Sunday.
- The day of the week.
- The month.

The "nth" weekday uses these values in the field designation:

1 - The first weekday of the month.

2 - The second weekday of the month

3 - The third weekday of the month.

4 - The fourth weekday of the month.

5 - The fifth weekday of the month.

-1 - The last weekday of the month.

The "day of the week" is indicated by the values 0 to 6:

0 - Sunday.

1 - Monday.

2 - Tuesday.

3 - Wednesday.

4 - Thursday.

5 - Friday.

6 - Saturday.

The month is indicated by numerical values:

1 - January.

2 - February.

...

12 - December.

All date rules must have all three fields (weekday, day of week, and month). Each field is separated by a semi-colon (';') character. No spaces or other characters are allowed in a date rule. Double quotes around the fields are not required.

Here are some examples of date rules:

"-1;5;9" = The last Friday in September.

"2;1;3" = The second Monday in March.

Field Descriptions

A description of the eleven fields in each row of the time zone database.

- ID: (Required). Contains the identifying label for the time zone region. A "region" is a longer string used to identify the time zone. An example would be America/Pacific for the Pacific coast time zone in America. Any string can suffice as long as it is unique within the time zone database.
- STD Abbreviation: (Required). This contains the abbreviation of the time zone. The STD means Standard Time Zone Designator. This is the familiar PST or EST, and is the standard time when daylight savings is not in effect. Abbreviations should consist only of upper case letters, while the name can be any descriptive string. Lower case abbreviations, if present, are capitalized by the system.
- STD Name: The STD means Standard Time Zone Designator. The STD Name is the longer version of the name for standard time. For PST this could be Pacific Standard Time.
- DST Abbreviation: This contains the abbreviation of the time zone. The DST Abbreviation is used to label daylight savings time. For example, in the PST time zone, this would be PDT. Abbreviations should consist only of upper case letters while the name may be any descriptive string. Lower case abbreviations, if present, are capitalized by the system. This entry is optional; time zones that do not use daylight savings time do not need the DST abbreviation and name.
- DST Name: The DST name is a longer version of the name of the time zone. For PDT this could be Pacific Daylight Savings Time. This entry is optional; time zones that do not use daylight savings time do not need the DST abbreviation and name. Please see below for further discussion of DST entries.
- GMT Offset: This is the number of hours added to Greenwich Mean Time to get the local time before any daylight savings adjustments are made. Some examples are: America/Pacific offset: -8:00 hours, Africa/Cairo offset: +2:00 hours.
- DST Adjustment: The amount of time added to the local time when daylight savings is in effect. This format must follow the length-of-time format described above. For the United States this is typically "1:00"
- DST Start Date Rule: Describes the day of the year in which the transition to daylight savings time takes place. See Date Rules" above for specific formatting rules.
- Start Time: The time of day to begin daylight savings time in 24 hour time format. The format must follow the length-of-time describe above. For time zones in the United States this is typically "+02:00" meaning daylight savings starts 2 hours past midnight.
- DST End Date Rule: Describes the day of the year in which daylight savings time ends and standard time begins again.

- End Time: The same as the Start Time, but for the ending date. For the United States this is typically "+02:00".

Optional Fields - Daylight Savings

Daylight savings time is optional. Many time zones do not use daylight savings time. The time zone software insists that daylight savings time either be wholly present or wholly absent. An error message is generated if this is not true. A missing field is indicated by simply using a comma to skip to the next field. Thus, for Japan Standard Time, there is no daylight savings time and the entry would be similar to:

ID - "Asia/Japan"

STD Abbr - "JST

STD Name - "Japan Standard Time

DST Abbrev - empty

DST Name - empty

GMT Offset - +09:00

DST Adjustment - empty

DST Start Rule - empty

DST Start Time - empty

DST End Date Rule - empty

DST End Time - empty

Warning! The Sybase CEP Time Zone Database stores only one set of information about each time zone. This means that when rules (for example, for daylight savings time) are updated, Sybase CEP knows only the current rules. Thus, for example, if in the year 2007 you change the starting date of daylight savings time from "the first Sunday in April" to "the second Sunday in March", then after you update the Sybase CEP time zone database, Sybase CEP will treat the second Sunday in March as the start of daylight savings time, even for years prior to 2007.

CCL Abbreviations

The fields in the time zone CSV database are used by the Sybase CEP software to link external time zone names to the proper time calculations. Within the time zone database, the following must be unique within the entire database:

- ID.
- STD Abbreviation.
- DST Abbreviation.

This uniqueness allows users to identify time zones in various ways. The identification applies to both input and output of time zones.

- ID maps to TZR, the Time Zone Region. This is output only.
- STD abbreviation maps to TZD, the Time Zone Designator. Use in the input or output.
- DST abbreviation maps to TZD as well. Use this in input or output.
- TZH:TZM displays as time zone hours and minutes from GMT. This is output only. While TZH and TZM are typically displayed together, they may be separated and placed however you desire.

Standard and Daylight savings time abbreviations can be used interchangeably. The Sybase CEP software can, if applicable, modify the time for daylight savings and interpret the appropriate STD or DST abbreviation. As in all date-time format strings, multiple uses of the same format code are illegal on input, but are legal in output formats. Thus the following is illegal because multiple TZD formats are illegal for input:

```
TO_TIMESTAMP("2002-06-18 PDT13:52:00.123456 PST,
   "YYYY-MM-DD TZDHH24:MI:SS.ff TZD");
```

An output request allows multiple format requests:

```
TO_STRING(ts, "YYYY-MM-DD TZDTZDTZD", PDT)
   yields: "2006-06-18 PDTPDTPDT
```

Index

.NET3/C#
Locating Files 291
.NET3/C# Examples
Compiling and Running 294

64-bit, Compiling C/C++ on
Compiling for 64-bit Microsoft Windows 102

A

a Project with .NET3/C#
Compiling and Starting a Project 296
a Project with C/C++
Compiling a CCL Project 210
a Project, with c8_client
Compile a Project or a Schema File 71
Absolute vs. Relative Stream URIs 366
Adaer, Feed Reader
RSS Feed Reader Adapter. 491
Adapers
Configuring Your Environment for SNMP Adapters 513
Adapter 511
Database: Poll From DB Input Adapter 414
Windows Event Logger Adapter 509
Adapter Definition Language 319
Adapter Definition Language (ADL) 319
Adapter, Get OIDs
SNMP Get OIDs Adapter 492
Adapter, Input
Tuple Descriptor File 472
Adapter, Java
The email: Java Email Output Adapter. 466
Adapter, Output
JDBC Output Adapter 473
Adapter, Poll from
Retrieving a Subset of Records 417
Adapter, Read from
Database: Read From DB Input Adapter 420
Adapter, Read from File
Binary: Read From Binary File Adapter. 391
Loop Count 403
Regular Expressions: Read From File Using a Regular Expression Adapter. 485
XML: Read From XML File 500
Adapter, Read from Socket
Comma-Separated Values (CSV): Read From CSV Socket Adapter. 409
Regular Expressions: Read From Socket Using a Regular Expression Adapter. 489
XML: Read from XML Socket 504
Adapter, Send
Email: Send Email Out Adapter (SMTP). 464
Adapter, Send V1 Traps
SNMP Send V1 Traps Adapter 496
Adapter, Send V2c Notifications
SNMP Send V2c Notifications 498
Adapter, Set
SNMP Set Adapter 495
Adapter, Write over HTTP
XML: Write XML Over HTTP Adapter. 508
Adapter, Write to
Database: Write to DB Output Adapter 423
Adapter, Write to File
Comma-Separated Values (CSV): Write To CSV File Adapter. 405
XML: Write To XML File 505
Adapter, Write to Socket
Comma-Separated Values (CSV): Write To CSV Socket Adapter. 413
XML: Write to XML Socket 508
adapters 13
Adapters
Create a New JMS Server and Deploy It. 380
Adapters, Environment
Setting Up the Environment for the JMS Adapters 380
Adapters, Environment for
Setting Up the Environment for Java Adapters 379
ADL
Adapter Definition Language (ADL) 319
Aggregate
User-Defined Aggregate Functions 240
Aggregate, User-Defined
User-Defined Aggregate Functions 240
Allocation with C/C++

Notes about Allocating and Deallocating Memory in In-process Code 110
and Adapters
Reading, Writing, and Converting Timestamps 384
and Container Events
Container Events 328
and Distributed Queries
URIs and Distributed Queries 366
and High Availability
URIs and High Availability 367
and Manager Events
Manager Events 329
and Persistence
Persistence 92
and URIs
URIs and High Availability 367
API for Managing
API Functions for Managing Session State 266
API Functions for Publishing Messages 266
API Interface 303, 313
API, C/C++
Schema API 118
APIs
Status APIs 36
apis used for in-process and out-of-process adapters 118
APIs Used for Out-of-process Adapters and Control Programs 117
Application
Status Information about a CCL Application 519
Application Components
Application Components 89
Application Development
Application Components 89
Arbitrary Predicates
Out-of-process Adapter 17
Asynchronously, in Java
Publishing Asynchronously 287
Asynchronously, with Java
Subscribing Asynchronously 288
Atom Feed Reader
Atom Feed Reader Input Adapter 389
Authentication
User Authentication via Pluggable Authentication Module (PAM) 341
Authentication with
User Authentication 271
Authentication, Creating
Creating Your Own Authentication Plugin 273

B

Binary Format
Binary Data Stream Format 373
Binding
Binding A Registered Query's Stream to An Existing Stream 32
Binding A Registered Query's Stream to An Existing Stream 32
BLOB Data with
Reading and Writing BLOB Data 379
Bundles 11
Working with Bundles 288, 298

C

C/C++
Overview 101
C/C++ API for
Data Types and Subroutines for UDFs and In-process Adapters 102
c/c++ data conversion functions 111
C/C++ on 64-bit Windows
Compiling for 64-bit Microsoft Windows 102
c8_compiler
Compiling Directly via the CCL Compiler 72
c8_server
Engine Control: Command-line Tools 67
c8-services.xml
The c8-services.xml File 59
c8client.h 117
CCL Stream Pairs
Status Information about CCL Stream Pairs 524
CclApplicationInfo
CclApplicationInfo Status Messages 330
CclApplicationInfo Status Messages
CclApplicationInfo Status Messages 330
CclCompilerInfo Status Messages
Status Information about CCL Compiler Settings 522
CclQueryInfo
CclQueryInfo Status Messages 332

CCLQueryInfo Status Messages
CclQueryInfo Status Messages 332
CclStreamPairInfo Status Messages
Status Information about CCL Stream Pairs 524
CclWorkspaceInfo Status Messages
Status Information about a Workspace 524
CEP 511
Comma-Separated Values (CSV): Read From CSV File Adapter. 391
communication 17
Compared to Client-Side
Client-side vs. Server-side Compilation 30
Compared to In-Process
In-process vs. Out-of-process Adapters 15
Compared to Out-of-Process
In-process vs. Out-of-process Adapters 15
Compared to RPCs
How to Choose Whether to Use an RPC or a UDF 47
Compared to Server-Side
Client-side vs. Server-side Compilation 30
Compared to User-Defined Function
How to Choose Whether to Use an RPC or a UDF 47
Compiler API
The Compiler API 210
Compiler Settings
Status Information about CCL Compiler Settings 522
Compiling
Compiling a UDF 256
Compiling an In-process Adapter 190
Compiling a Project with
Compile a Project or a Schema File 71
Compiling a CCL Project 210
Compiling and Starting a Project 296
Compiling a Sybase CEP RPC Plugin 56
Compiling a UDF 256
Compiling a UDF and Putting It in the Correct Directory 256
Compiling an RPC Plugin and Putting It in the Correct Directory 267
Compiling and Running 294
Compiling and Running a Project
Compiling and Running a Project 81
Compiling and Starting a Project with
Compiling and Starting a Project 285
Compiling and Starting with C/C++
Compiling a CCL Project 210
Compiling and Starting with c8_client
Compiling and Running a Project 81
Compiling and Starting, with .NET3/C#
Compiling and Starting a Project 296
Compiling and Starting, with Java
Compiling and Starting a Project 285
Compiling Directly via the CCL Compiler 72
Compiling Errors
Error Messages When Compiling Java-Language Adapters 352
Compiling on
Compiling for 64-bit Microsoft Windows 102
Compiling on 64-bit Windows
Compiling for 64-bit Microsoft Windows 102
Compiling with c8_client
Compile a Project or a Schema File 71
Compiling with c8_compiler
Compiling Directly via the CCL Compiler 72
components 57
Configuring and Setting Up Your JMS Server 380
Configuring Your System to Find ADL Files 324
Connecting to a Stream Over
Connecting to Streams over a Network 369
Connecting to Over a Network
Connecting to Streams over a Network 369
Connectivity Instructions for UNIX-like Operating Systems 472
Container
Status Information about a Container 525
ContainerInfo
ContainerInfo Status Messages 332
ContainerInfo Status Messages
ContainerInfo Status Messages 332
Control: Compile/Start/Stop/Status 210
Create a New JMS Server and Deploy It. 380
create a project and its associated schema files 70
Create a Workspace with
Create a Workspace on the Server 71
Creating
Creating Your Own Authentication Plugin 273
Creating a URI with
Creating a URI 281, 293
Creating an Engine Client in
Creating an Engine Client 281
Creating an Engine Client with
Creating an Engine Client 292
Creating an In-process Adapter in C/C++ 157

Creating an Out-of-process Adapter in C/C++ 141
Creating in .NET3/C#
    Creating an Engine Client 292
Creating in Java
    Creating an Engine Client 281
creating streams and CCL statements from inside a program 24
creating your own stream adapter overview 16
creating your own stream adapter: overview 16
Creating, with .NET3/C#
    Creating a URI 293
Creating, with c8_client
    Create a Workspace on the Server 71
Creating, with Java
    Creating a URI 281
CSV Format
    CSV Data Stream Format 373

D

Data Stream Format
    Binary Data Stream Format 373
    CSV Data Stream Format 373
    XML Data Stream Format 375
Data Stream Formats
    Data Stream Formats 373
Data Stream URI 369
Data Type Mapping to
    Datatype Mappings 347
Data Types
    Datatype Mappings 347
Data Types and Subroutines for UDFs and In-process Adapters 102
Data with Adapters
    Reading and Writing BLOB Data 379
Database, Poll from
    Retrieving a Subset of Records 417
Database: Poll From DB Input Adapter 414
Daylight Saving Time and the Sybase CEP Time Zone Database 567
Definition
    Message / Row 7
    Query / Statement 21
    Schema 8
    Streams and Adapters 14
    What Is a URI and What Is It Used For? 365
Definition Language for
    Adapter Definition Language (ADL) 319
Determining
    How to Find the URI of a Stream 367
Determining for a Stream
    Acquiring the Address (URI) of a Stream 150
Determining URI of
    How to Find the URI of a Stream 367
Dynamically Registering Queries and Streams 26

E

Engine Control
    Controlling the Engine 284
Engine Control with
    Control: Compile/Start/Stop/Status 210
    Controlling the Engine 284, 295
Engine Control: Command-line Tools 67
Examining Schemas with
    Examining Schemas 286, 297
Examining with .NET3/C#
    Examining Schemas 297
Examining, with Java
    Examining Schemas 286
Examples
    Using the Examples 280, 291
Examples, Compiling and Running
    Compiling and Running 282, 294
Execution 183

F

Filter Expressions
    Out-of-process Adapter 17
for Guaranteed Delivery, with .NET3/C#
    Publishing 97
from the Command-Line
    Engine Control: Command-line Tools 67
Functions for Accessing Configuration Information 265
Functions Primarily for Aggregator UDFs 254

G

Ganglia Input
    Ganglia Input Adapter 469
Ganglia Input Adapter 469
Generic HTTP and SOAP Plugins 52
Guaranteed Delivery Mechanisms 93
Guaranteed Delivery with

Guaranteed Delivery with the Sybase CEP .NET SDK 96
Publishing 97
Guaranteed Delivery with the Sybase CEP .NET3 SDK 96
guaranteed delivery with the sybase cep c/c++ sdk 94
Guaranteeing Message Delivery 289, 299

H

htpassword
User Authentication htpasswd Plugin 338
htpassword Plug-In
User Authentication htpasswd Plugin 338
HTTP
HTTP and SOAP Plugin Preferences 361
HTTP and SOAP Plugin Preferences 361
HTTP URL
Working with URIs 289

I

Implementing
Guaranteed Processing Implementation 90
implementing guaranteed processing 89
implementing with c++ 94
Implementing, with .NET3/C#
Guaranteeing Message Delivery 299
Importing New Adapters 326
in .NET3/C# SDK
Working with Bundles 298
In-Process
In-process Adapter 19
In-Process Adapter
Compiling an In-process Adapter 190
In-process Adapter 19
In-process Adapter API 161
In-Process Adapter API
In-process Adapter API 161
In-Process Adapter with
Creating an In-process Adapter in C/C++ 157
In-Process Adapter, Multi-Stream
Multi-Stream In-Process Adapters 196
In-Process Compared to Out-of-Process
In-process vs. Out-of-process Adapters 15
In-process Output Adapters 181
In-Process, API for
In-process Adapter API 161
In-Process, Compiling
Compiling an In-process Adapter 190
In-Process, Writing
Initialization 182
Infiniband support 372
Input Adapter
Atom Feed Reader Input Adapter 389
Ganglia Input Adapter 469
Installation 510
introduction 23

J

Java
Error Messages When Compiling Java-Language Adapters 352
Java Examples
Compiling and Running 282
Java, and Errors
Error Messages When Compiling Java-Language Adapters 352
Java, Environment for
Setting Up the Environment for Java Adapters 379
JDBC Input
Tuple Descriptor File 472
JDBC Input Adapter 469
JDBC Output
JDBC Output Adapter 473
JMS
Setting Up the Environment for the JMS Adapters 380
JMS Adapter 476

L

LDAP
User Authentication LDAP plugin 340
LDAP Plug-In for
User Authentication LDAP plugin 340
Lengths of Time 570
Line Continuation Character 403
Locating Files 279
Logical vs. Physical Stream URIs 366

M

Manager
Status Information about a Manager 566
managerInfo Status Messages
managerInfo Status Messages 333
managerInfor
managerInfo Status Messages 333
Mapping to Database Types
Datatype Mappings 347
Memory Allocation
Notes about Allocating and Deallocating Memory in In-process Code 110
memory management api 109
Memory Management API
Memory Management API 109
Message API
Message API 121

O

On Client Compared to On Server
Client-side vs. Server-side Compilation 30
Out-of-Process Adapter with
APIs Used for Out-of-process Adapters and Control Programs 117
Out-of-Process, Writing
Out-of-process Adapter 17
Out-of-Process, Writing in C/C++
c8client.h 117
overview 89

P

Perl
Prerequisites 303
Persistence
Persistence 92
Persistent State
Session State Information and Persistent State Information 159
Plug-In
Generic HTTP and SOAP Plugins 52
Server Plugins 327
The RPC Plugin 51
User Authentication LDAP plugin 340
Plug-In, API
RPC Plugin API 264
Plug-In, compiling
Compiling an RPC Plugin 268
Plug-In, for CSV Files
RPC Plugin for CSV Files 53
Plug-Ins
The User Authentication Plugins 337
Plugin Configuration 276
Poll from Database
Database: Poll From DB Input Adapter 414
Predicates
Out-of-process Adapter 17
Prerequisites 380
Printing a Schema with
Printing the Schema 185
Printing, with C/C++
Printing the Schema 185
Project
Status Information about a CCL Application 519
Project Status with
Get Status of an Executing Project 82
Project, Retrieving with c8_client
Get Status of an Executing Project 82
Properties
Settings for Projects, Modules, and Streams 91
Public
Public Windows 65
Public Windows 65
Public Windows, Querying with
Querying a Public Window 262
Public, Querying with .NET3/C#
Querying a Public Window 300
Public, Querying with C/C++
Querying a Public Window 262
Public, Querying with Java
Querying a Public Window 290
Publishing 97
Publishing Asynchronously with
Publishing Asynchronously 287
Publishing for an In-Process Adapter 94
Publishing to a Stream with
Publishing to a Data Stream 371
Publishing with
Publishing Data to a Server 82
Publishing to a Stream 283, 294
Publishing, In-Process Adapter
Publishing for an In-Process Adapter 94
Publishing, Out-of-Process Adapter
Publishing for an Out-of-Process Adapter 95
Publishing, with C#/.NET3

Publishing 97
Publishing, with Java
Publishing Asynchronously 287

Q

Query
Status Information for a CCL Query 523
Querying a Public Window with
Querying a Public Window 290, 300
Querying, with .NET3/C#
Querying a Public Window 290
Querying, with C/C++
Querying a Public Window 262
Querying, with Java
Querying a Public Window 300

R

Random Message Generator
Random Tuples Generator Adapter 480
Random Message Generator Adapter
Random Tuples Generator Adapter 480
RAP Adapter
SybaseRAPAdapter 482–485
Read from Binary File
Binary: Read From Binary File Adapter. 391
Read from CSV File
Loop Count 403
Read from CSV Socket
Comma-Separated Values (CSV): Read From CSV Socket Adapter. 409
Read from Database
Database: Read From DB Input Adapter 420
Read from XML File
XML: Read From XML File 500
Read from XML Socket
XML: Read from XML Socket 504
reading and writing BLOBs on external database servers 60
Reading from a Stream
Reading Data from a Stream 293
Reading from a Stream with
Reading Data from a Stream 282
Registering
Guidelines 31
Registering a Parameterized Query with
Registering a Query with Parameters 300
Registering a Project with
Registering a Project via the c8_client Program 80
Registering a Query with
Registering a Query 285, 296
Setting Up Dynamic Queries and Streams with the C/C++ SDK 197
Registering Queries
Guidelines 31
Registering with .NET3/C#
Registering a Query with Parameters 300
Registering with C/C++
Setting Up Dynamic Queries and Streams with the C/C++ SDK 197
Registering with c8_client
Registering a Project via the c8_client Program 80
Registering with Java
Registering a Query 285
Regular Expression Read from File
Regular Expressions: Read From File Using a Regular Expression Adapter. 485
Regular Expression Read from Socket
Regular Expressions: Read From Socket Using a Regular Expression Adapter. 489
Remote Database
The Components 57
remote database queries 56
Remote Database Query
The Components 57
remote procedures 47
Remote Queries
The Components 57
remote requests, synchronization, and performance 64
Requirements 41
Retrieving Server Status with
Retrieving Server Status 287, 298
Retrieving Status with .NET3/C#
Retrieving Server Status 298
Retrieving Status with c8_client
Get Status of an Executing Project 82
RPC
RPC Plugins 264
What Is an RPC? 48
RPC Plug-In
Compiling an RPC Plugin 268
RPC Plug-In API
RPC Plugin API 264
RPC Plugins 264
RPC, compiling
Compiling an RPC Plugin 268

RPC, for CSV Files
RPC Plugin for CSV Files 53
RPC, Writing
Writing Your Own Sybase CEP RPC Plugin 55
RSS Feed Reader
RSS Feed Reader Adapter. 491
Running the Installation Script 308

S

Schema API
Schema API 118
SDK
Error Messages When Compiling Java-Language Adapters 352
Locating Files 291
Overview 101
Prerequisites 303
Send Email
Email: Send Email Out Adapter (SMTP). 464
Server API
Server API 173
Session State
Session State Information and Persistent State Information 159
Session State API
API Functions for Managing Session State 266
Session State Information and Persistent State Information 159
Session States vs. Persistent States 174
Setting Up Your Environment 279
Settings for Projects, Modules, and Streams 91
Shutdown 184
SMTP
Email: Send Email Out Adapter (SMTP). 464
SNMP
Configuring Your Environment for SNMP Adapters 513
SNMP Get OIDs
SNMP Get OIDs Adapter 492
SNMP Send V1 Traps
SNMP Send V1 Traps Adapter 496
SNMP Send V2c Notifications
SNMP Send V2c Notifications 498
SNMP Set
SNMP Set Adapter 495
SOAP
HTTP and SOAP Plugin Preferences 361
Start with
Start with Clean Slate 92
Start with Clean Slate 92
Start with Clean Slate 92
Starting a Project
Execute the Project 79
Starting and Stopping
Starting and Stopping Adapters 20
Starting with c8_client
Execute the Project 79
Status API 220
API Functions for Reading Runtime Status 267
Status API 220
Status Information 34
Status Information 519
Status Information about a Container 525
Status Information about a Workspace 524
Status Information for a CCL Query 523
Status Information About
Status Information about a CCL Application 519
Status Information about a CCL Application 519
Status Information about a Container 525
Status Information about a Manager 566
Status Information for a CCL Query 523
Status Stream
Status Streams 34
stop the sybase cep server 86
Stopping a Query
Stopping A Query 33
Stopping, through SDK
Stopping A Query 33
Stream URIs 365
Subscriber 315
Subscribing 98
Subscribing Asynchronously with
Subscribing Asynchronously 288
Subscribing for an In-Process Adapter 95
Subscribing to a Stream 282
Subscribing to a Stream with
Examining Example 1: Subscribing to a Stream 281
Subscribing to a Data Stream 369
Subscribing to a Stream 293
Subscribing to with HTTP
Subscribing to a Data Stream 369
Subscribing, in .NET3/C#

Subscribing 98
Subscribing, in C/C++
Subscribing for an Out-of-Process Adapter 96
Sybase CEP Drivers 517
Sybase CEP Engine Settings 91
Sybase RAP
SybaseRAPAdapter 482–485

T

The CCL Statement 50, 58
The Components of an In-process Adapter 158
The email, Java
The email: Java Email Output Adapter. 466
The RPC Plugin 51
The Sybase CEP Server 51, 59
The User Authentication Plugins 337
Time Zone Database
Field Descriptions 572
Timestamp Base 401
Timestamps and
Reading, Writing, and Converting Timestamps 384
Title Rows and the Hidden Row Timestamp Column 400
Tracer Message API
Tracer Message API 233
Troubleshooting 33, 197, 210, 311
General Tips 351
Tuple
Tuple 314
Tuple Descriptor
Tuple Descriptor 8
Tuples, Working with in
Working with Tuples 286, 297

U

UDFs: Packing and Unpacking Parameter Values 235
UDFs: Requirements and Example 235
UDFs: Summary 261
UDFs: XML Signatures 242
URI, Determining
Acquiring the Address (URI) of a Stream 150
URIs
Types of URIs 365
Usage 510
User Authentication
The User Authentication Plugins 337
User Authentication via Pluggable Authentication Module (PAM) 341
User Authentication from Inside SDKs 38
User Authentication Plug-In
User Authentication htpasswd Plugin 338
User Authentication via Pluggable Authentication Module (PAM) 341
User Credentials API
User Credentials API 272
User-Defined Aggregate Functions 41, 240
User-Defined Function
Compiling a UDF and Putting It in the Correct Directory 256
User-Defined Function API
Data Types and Subroutines for UDFs and In-process Adapters 102
User-Defined Functions 10, 234
Plugins 39
User-Defined, in C
User-Defined Functions 39

V

Value Types with
Exploring Value Types 286, 297
Variations of
Variations of Guaranteed Processing 99

W

Windows and Expiring Messages 10
Windows Event Log 511
Windows Event Logger
Windows Event Logger Adapter 509
with .NET3/C#
Controlling the Engine 295
Guaranteed Delivery with the Sybase CEP .NET3 SDK 96
Publishing to a Stream 294
Subscribing to a Stream 293
with C/C++
Control: Compile/Start/Stop/Status 210
User Authentication 271
with c8_client
Publishing Data to a Server 82
with c8_compiler
Compiling Directly via the CCL Compiler 72
with Guaranteed Delivery, in .NET3/C#
Subscribing 98
with Guaranteed Delivery, in C/C++

Subscribing for an Out-of-Process Adapter 96
with HTTP
Publishing to a Data Stream 371
Subscribing to a Data Stream 369
with Java
Controlling the Engine 284
Examining Example 1: Subscribing to a Stream 281
Publishing to a Stream 283
Working with
Working with URIs 289
Working with Parallel Queries 290, 301
Working with URIs 300
Working with, in .NET3/C#
Working with Tuples 297
Working with, in Java
Working with Tuples 286
Workspace
Status Information about a Workspace 524
Workspace 22
Write to Binary File
Binary: Write To Binary File Adapter. 391
Write to Binary File Adapter
Binary: Write To Binary File Adapter. 391
Write to CSV File
Comma-Separated Values (CSV): Write To CSV File Adapter. 405
Write to CSV Socket
Comma-Separated Values (CSV): Write To CSV Socket Adapter. 413
Write to Database
Database: Write to DB Output Adapter 423
Write to XML File
XML: Write To XML File 505
Write to XML Socket
XML: Write to XML Socket 508
Write XML over HTTP
XML: Write XML Over HTTP Adapter. 508
Writing
Initialization 182
Out-of-process Adapter 17
Plugins 44
Signatures of User Functions 176
writing an adapter for guaranteed processing 93
Writing in C/C++
c8client.h 117
Writing Your Own Sybase CEP RPC Plugin 55
Writing, in C/C++
User-Defined Functions 39
Writing, Multi-Stream
Multi-Stream In-Process Adapters 196

X

XML Format
XML Data Stream Format 375
XML Signature for
UDFs: XML Signatures 242
XML Signatures 42