SYBASE®

Coral8 Integration Guide

DOCUMENT ID: DC01030-01-0200-02

LAST REVISED: May 2009

Table of Contents

Preface

This document covers a wide variety of topics that are related to "interfacing Coral8 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 Coral8 Server, e.g. 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 Coral8, as well as for software architects who plan to use Coral8 as part of a larger system and need an overview of how Coral8 interfaces with the outside world.

This guide assumes that you have already read the Programmer's Guide and have general familiarity with the Coral8 Engine.

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

How to Use This Guide

This manual is not intended to be read linearly.

The first several chapters of this guide provide conceptual overviews of topics such as adapters, RPC (Remote Procedure Calls), and user-defined functions (UDFs) written in C/C++. The next several chapters explain how to write software to do these tasks (e.g. write an adapter) using the SDKs (Software Development Kits) that Coral8 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 "coral8/server" directory. On Microsoft Windows systems, this directory is typically:

```
C:\Program Files\Coral8\Server\sdk
```

On UNIX-like operating systems, this directory is typically

```
/home/<userID>/coral8/server/sdk
```

Under this directory, you will find a subdirectory for each of the languages for which Coral8 supplies an SDK (including C/C++, Java, .NET, Python, and Perl).

Integrating Coral8 with External Systems

This document covers a wide variety of topics that are related to interfacing Coral8 with external systems.

Topics in this manual include:

- Data input and output
    - Coral8 *adapters*: adapters are software that convert data from external formats to the Coral8 Server's native format and vice versa. Coral8 provides a set of adapters that convert between the server's format and several common data formats. Coral8 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.
- Server Control
    - Starting and stopping execution of a project (query module), getting the status of an executing project, etc.
    - Using command-line tools (such as the Coral8 CCL compiler) from outside Coral8 Studio.
- Extending the Coral8 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 (e.g. such as a server process shutting down)
    - Remote Procedure Calls (RPCs)
- Accessing Coral8 streams directly over the network
- Embedding the server as a library inside another process

The table below shows some of the capabilities of each of the Coral8 SDKs. 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.

SDK	In Process Adapters	Out of Process Adapters	Dynamic Queries	User Defined Functions	CSV Message format	XML Message Format	Binary Message Format	Operations on Message "Bundles"
C/C++	+	+	+	+	+	+	+	+
Java		+	+			+	+	+
.NET		+	+		+			

Perl		+			+			
Python		+			+			

Coral8 uses the term "Coral8 Engine" to refer to the entire product, and uses the term "Coral8 Server" to refer to the program that processes queries and generates output. The term "Coral8 Engine" thus includes:

- Coral8 Server
- Coral8 Studio, which is a GUI tool that helps you create and edit queries, streams, etc.
- Coral8 SDKs, which help you create adapters, user-defined functions, and separate applications that act as "clients" of Coral8 Server
- Coral8 command-line utilities, which do tasks such as compiling queries that are written in the CCL query language

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

Coral8 Integration and Enterprise Software Architecture

Coral8 Engine can be used as a largely stand-alone product or as part of a larger system. Coral8 Engine allows you to do the following to integrate the Engine into a larger system:

- Get data into and out of the Coral8 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, e.g. by adding new functions that can be called from CCL queries.
- Get data from, or send data to, a database server.
- Embed Coral8 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 Coral8 Engine can connect to the outside world (e.g. 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 Coral8 Integration

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

- which languages (e.g. C, Java, .NET, Perl) Coral8 provides SDKs for;

- which functionality is in each SDK, e.g.
    - input/output,
    - control,
    - embedability,
    - User-Defined Functions

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 Coral8 Engine is capable of doing. The remaining chapters focus on the "how", e.g. 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 (e.g. getting data in and out of the server), then you may skip the detailed "how to" sections for other topics.

Other Interfaces to Coral8

Coral8 Server also supports other useful interfaces. For example, it provides a status stream that allows you to monitor servers and queries. It also provides an ability to control the Coral8 Server with an API based on SOAP (Simple Object Access Protocol).

Other Documents that You May Find Helpful

If you are interested in running multiple copies of the server, either to enhance reliability by having backup servers ("High Availability") or to enhance performance by doing more parallel processing ("parallel queries"), see the Administration Guide.

If you want to know how to read data from (or write data to) database servers and use that data in Coral8 Server (without writing your own adapter), see the discussion of database subqueries and the Database Statement in the Coral8 CCL Reference.

Data Streams and Messages

This chapter explains streams, messages, and other key concepts.

To create adapters (which send data to, or read data from, the server), you need a good understanding of data streams and messages. We'll start by defining key terms that we use. If you have already read the Programmer's Guide, some of these terms will be familiar to you. We list them here so that we can "build up" from simple concepts to more complex concepts.

Definitions

Field

A field holds a single piece of data, such as a salary, or a person's last name, or a timestamp (e.g. 2006-07-01 15:00:00.000000). For some data types, such as XML, the content may be more "complex", but to Coral8 Server it's still a single piece of data.

Because database tables and Coral8 windows can be thought of as arranging data in rows and columns, we often casually 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" or 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 (defined below).

Row Timestamp

In a Coral8 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 Coral8 TIMESTAMP data type, which means that the time information is stored with a precision of 1 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 thus about the schema of the rows in the stream. Adapters written using certain Coral8 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 (usually called just "stream") "carries" rows from one place to another, e.g. 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 (Uniform Resource Identifier), which serves as "address" to enable 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.coral8.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 Coral8 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* for a more detailed discussion of CCL vs. HTTP 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, i.e. 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, then they can simply insert the rows in the proper order. If the publishers do not synchronize with each other, then set the appropriate stream properties ("Maximum Delay", "Messages can come out of order", and "Maximum out of order delay") to tell the Coral8 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 CCL Programmer's Guide.

The most common way to publish to, or subscribe to, a stream is to create an adapter using one of the Coral8 SDKs.

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

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

Windows

If you have already read the Programmer's Guide, then you know that 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 allows you to relate a new row to rows that arrived earlier. It also allows you to perform aggregate functions (e.g. SUM(), AVG(), etc.) 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 accumulates messages indefinitely (i.e. is defined with a KEEP ALL clause), rows from the window will eventually expire and be deleted from the window.

User-Defined Functions

The CCL language contains many built in functions, such as SQRT(), AVG(), etc. You can write your own functions and call them from CCL; we call these functions 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. Here in the Integration Guide, we describe only C/C++ UDFs. For information about the CCL CREATE FUNCTION statement, see the Reference Guide.

C/C++ UDFs are described in more detail later in this manual; see *Coral8 C/C++ SDK*.

Windows and Expiring Messages

Advanced Topic: Expiring Messages

When rows are deleted from a window, some queries may need to be re-evaluated. For example, 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 when the row arrives, and one when the row expires.

Let's look at an example. First, let's create two windows and a set of queries in which 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 should (and 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 read rows from ("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 in the C/C++ SDK chapter.

Bundles

Advanced Topic

So far, we have described Coral8 streams as carrying one row at a time. However, there are exceptions to this. In some cases, a particular "event" (arrival of a row) may cause multiple output rows, and those rows may be grouped into a "bundle". A bundle is a set of 2 or more data messages that are conceptually part of the same operation (e.g. the output of a join, OUTPUT EVERY, etc.) and are thus grouped together. Your output adapter may see bundles and has the opportunity to process bundles differently from individual messages.

(Note that bundles apply only to adapters, not UDFs.)

Here are two situations in which bundles may occur.

- Suppose that 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).
- As a second example, consider what happens 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.

Note that 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 I have 100 stock transactions and I calculate the average price, I get only 1 output (the average price). (If I do the calculation again later, I'll get another output, of course, but for any single arriving row, I do the calculation once on the entire window, with all its rows, and generate a single output.) If I perform an operation that creates a bundle, that bundle may have multiple rows, not just 1 row, 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.

Note that 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.

Note also that a bundle may contain any type of message (e.g. 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 (e.g. the Coral8 Java SDK vs. the Coral8 C/C++ SDK) and are discussed in more detail later.

SDK

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

Coral8 supplies SDKs for various programming languages, including C/C++, Java, .NET. This document contains a chapter about each of these SDKs.

Adapters

This chapter provides an overview of adapters.

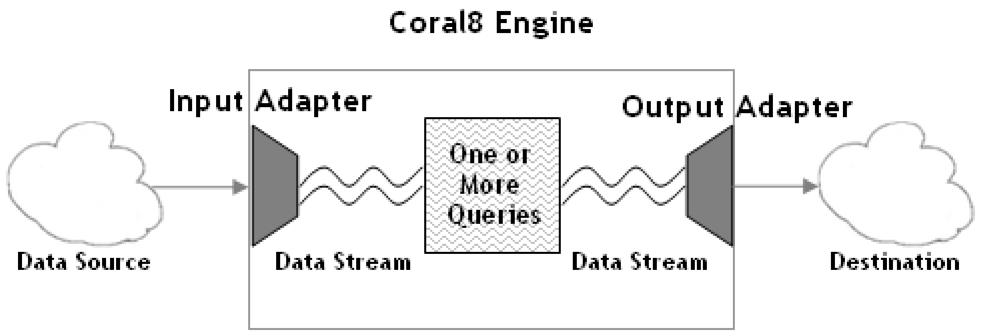
Streams and Adapters

An *input adapter* reads data from an external data source, and puts the data into a Coral8 stream. The input may be read from a file, a socket, a database server, an RSS feed, etc.

The data is then used in queries and the output of those queries is written to an output stream.

An *output adapter* reads data from the Coral8 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, etc.

The basic flow of data in a Coral8 Engine is thus:



Coral8 provides adapters that will read data from an external data source and write it to a Coral8 stream (or read from a Coral8 stream and write to an external data destination). For example, the ReadFromCSVFile adapter (supplied by Coral8) reads data from a file of Comma-Separated Values (CSV) and publishes the data to a stream. The Coral8 Server does not directly read data from the input file; the server reads only from data streams. The adapters supplied by Coral8 are described in *Coral8 Adapters*.

Some input and output adapters supplied by Coral8 can be used from inside Coral8 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 Coral8 will fit your needs, you may write your own adapter. Your adapter, like a Coral8-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 Coral8 Server.

Coral8 provides SDKs (Software Development Kits) that allow you to write input and output adapters in the following languages:

- C/C++
- Java
- .NET (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 Coral8 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.

(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 server when the server starts the corresponding project (query module). The adapter is recognized by Coral8 Studio, and from inside 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 Coral8 Server.

Out-of-process adapters deliver rows more slowly than in-process adapters do, are not recognized by 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 Coral8 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	Generally slower.

	the adapter with less overhead.	
Started by	Server	User
Recognized by Coral8 Studio	Yes	No
Can a crash in the adapter cause Coral8 Server to crash	Yes	No
Can run on a different computer than Coral8 Server	No	Yes
Languages	C/C++	C/C++ Java .NET Python Perl

Coral8 Adapters vs. User-written Adapters

Coral8 supplies several input and output adapters. (See *Coral8 Adapters* for a list.)

If these do not meet your needs, you may write your own adapters. Coral8 provides several SDKs to allow you to write an adapter. These SDKs include:

- C/C++
- Java
- .NET
- 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 Coral8.

Key Tasks: Conversion and Communication

Adapters perform two key tasks: *conversion* and *communication*.

An input adapter reads data from a source that is external to the Coral8 Engine (e.g. 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 Coral8 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, etc.

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

Your adapter will communicate by sending data to (or reading data from) a stream.

Out-of-process Adapter

If your adapter is an out-of-process adapter, then the adapter must do the following 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 Coral8) 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 Coral8 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 *Coral8 CCL Reference*. 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

If your adapter is an in-process adapter, it will communicate with the server via function calls.

Input Adapter Algorithm (Out-of-process)

This shows the algorithm of a typical out-of-process input adapter.

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.

Naturally, 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.

Output Adapter Algorithm (Out-of-process)

This shows the algorithm of a typical out-of-process output adapter.

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.

Adapter APIs

Coral8 provides APIs (Application Programmer Interfaces) that allow you to implement in-process and out-of-process adapters. APIs are available for:

1. In-process adapters in C/C++
2. Out-of-process adapters in C/C++
3. Out-of-process adapters in Java
4. Out-of-process adapters in .NET
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 (i.e 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). Thus we recommend that you 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: Overview

This chapter gives an overview of various "control" tasks, including:

- Create a Workspace in which you can load a query module.
- Delete a workspace
- Start the execution of a CCL program
- Dynamically add ("register") queries and streams
- Stop the execution of a CCL program
- Monitor servers and queries

These can be done via Studio, via command-line tools, and via an API in certain Coral8 SDKs. Before we explain how to do these tasks, we will define some terms and explain some concepts.

Definitions

This section contains definitions of key terms, including:

- Query / Statement
- Project / Query Module
- Workspace

Query / Statement

Coral8 queries are written in CCL, the Continuous Computation Language, which is based on the SQL language used in many database servers. This manual generally uses the terms "query" and "statement" synonymously.

Project / Query Module

CCL statements are stored in files with the extension ".ccl". A .ccl file may contain zero or more CCL statements, along with comments. 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, and thus 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 (e.g. "_1", "_2", etc.) each time that you load an additional copy of A into B.)

⚠ Note that since 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 a program can be run, it must be loaded into a workspace. Workspaces allow you to create groups in which all query modules are related in some way. The grouping is up to you. For example, you might create the groups based on projects, or based on individual developers, or based on some other criteria that you choose.

⚠ If more than one person loads the same query module into Studio, then makes changes and 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 Coral8 Studio to do tasks such as create a project, compile it, load it, start running it, stop running it, etc. These tasks can also be done by command-line utilities. Many of these tasks can also be done from inside a program by calling API functions provided by some of the Coral8 SDKs.

The following environments provide some or all of the interface functionality needed to deploy applications on Coral8 without using Coral8 Studio:

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

- Python SDK

A Note About ADL Files

Before compiling, the compiler reads .adl (Adapter Definition Language) files, which may be provided by Coral8 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* for details), then 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.

Commands

Most developers who build Coral8 applications will use the Coral8 Studio to create and execute query modules on Coral8 Server. Coral8 Studio provides an easy-to-use graphical user interface for editing, debugging and testing queries. Coral8 Studio automatically handles most of the details of interfacing the various components of the Coral8 Engine for the developer.

However, some developers may need to create and deploy their Coral8 applications from either another development environment (such as a scripting or programming language) or directly from the command line. Although these approaches require more detailed knowledge of the interfaces between the Coral8 components, the flexibility that is provided may offset this modest disadvantage.

In order to deploy a Coral8 application, you must be able to do the following things:

- Start Coral8 Server
- Access a running Coral8 Server
- Create a workspace on the Server
- Create a query module and its associated schema files
- Compile the query module using the Coral8 Compiler
- Execute the query module on the Server
- Stop the query module on the Server
- Stop the Server and clean up its resources

These tasks are executed using programs described later in this manual. Some Coral8 SDKs include function 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. For example, you might want to write a Java program that can dynamically create streams and queries on those streams.

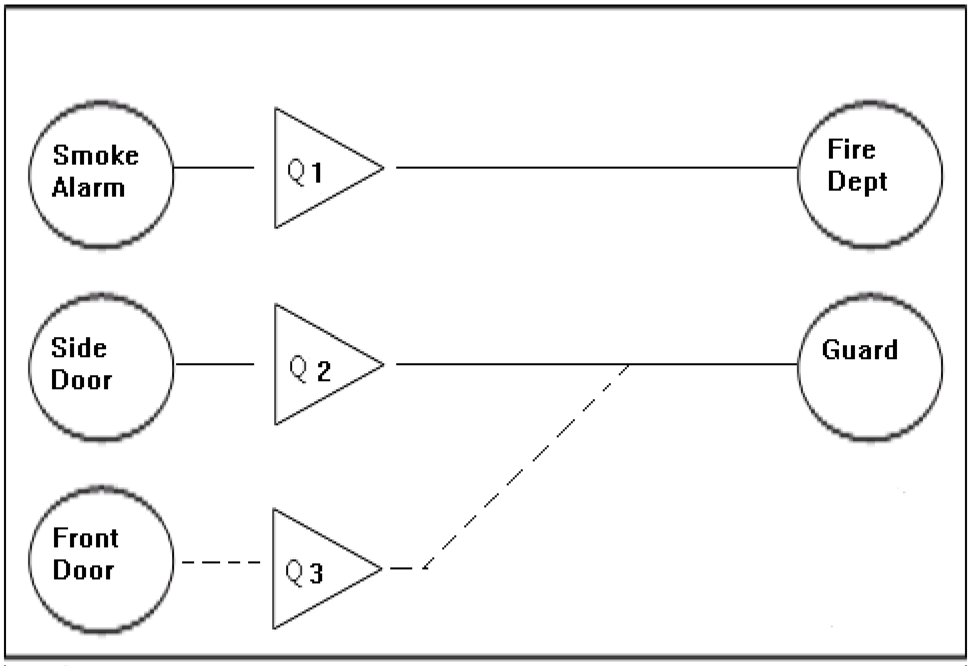
Motivation: When to Dynamically Create Queries and Streams

The ability to dynamically run and stop CCL statements and streams is useful if you have a system that is changing. For example, consider an application that monitors the security of a building. The input events come from door sensors, key-card readers, smoke detectors, etc. The output events include alerts to the security personnel, the fire department, etc. You need different queries on work days than on non-work days (e.g. holidays and weekends). Some queries need to be run continuously all day every day, while others should be activated and deactivated by software (automatically or via user intervention) based on the time of day and the day of the week.

Our building security application might contain the following:

- Smoke detector alerts. We monitor the smoke detectors and we 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, we send an alert to the on-site security guards if any "side doors" (any doors other than the main entrances) are opened AND a valid electronic badge was not used to open the door. (We require that people use their badge to exit, as well as to enter, outside normal working hours.)
- Front-door alerts. During non-working hours, we send an alert to the on-site security guards if a front door is opened AND a valid electronic badge was not used to open the door. During working hours, we do not send alerts when the front door is opened.

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



We would like the Smoke Alarm input stream, the Fire Department output stream, and query Q1 to be running all the time. We will use Coral8 Studio to create a project that contains these, and then we will start that project running and leave it running indefinitely.

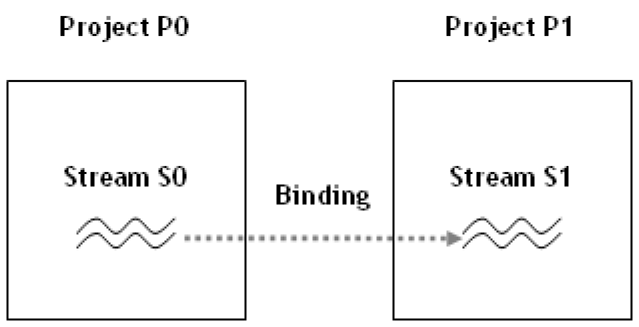
We would also like the Side Door input stream, the Guard output stream, and query Q2 to be running all the time. Therefore, these will be left running indefinitely.

We would like the Front Door input stream and the query Q3 to be running only outside of normal working hours. We will use a separate program to register them (create and run them) at appropriate times. We will stop running them during periods when they are not needed.

Background

As you know, you can use Studio to create a project (query module) that contains CCL statements and streams.

A CCL statement in one project (e.g. project "P1") may operate on data produced in another project (e.g. "P0") and send data to yet another project (e.g. "P2"). When you use Studio to set up data transfers between projects, you "bind" a stream in one project to a stream in the other project. Binding one stream to another means connecting the two streams. 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 not only from inside Coral8 Studio, but also from inside a program that you write. We call this "registering a query", and conceptually it is not much different from creating a project in Coral8 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:

- Coral8 C/C++ SDK
- Coral8 Java SDK
- Coral8 .NET SDK

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

Dynamically Registering Queries and Streams

Registering a query includes:

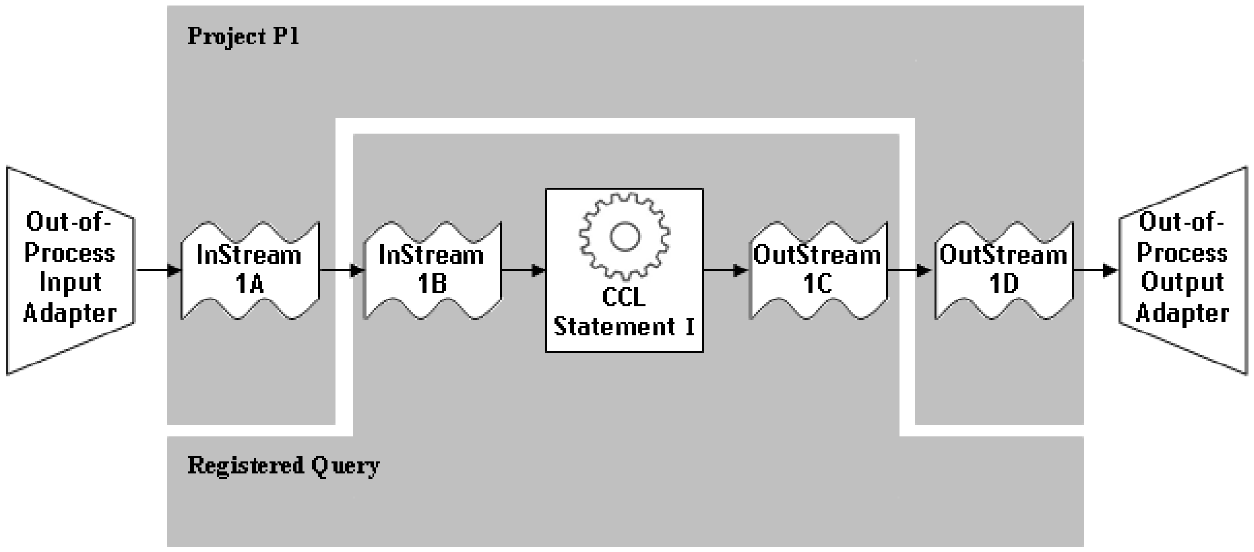
- creating the streams specified in the registered query;
- creating the statements specified in the registered query;
- compiling the CCL statements (note that, when we refer to "registering a query", the word "query" refers to zero or more CCL statements, not necessarily exactly 1 statement);
- loading 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 (i.e. to the URIs of those streams), or creating new streams;
- 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.

Note that 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 0, 1, or more CCL statements, and may contain 1 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:

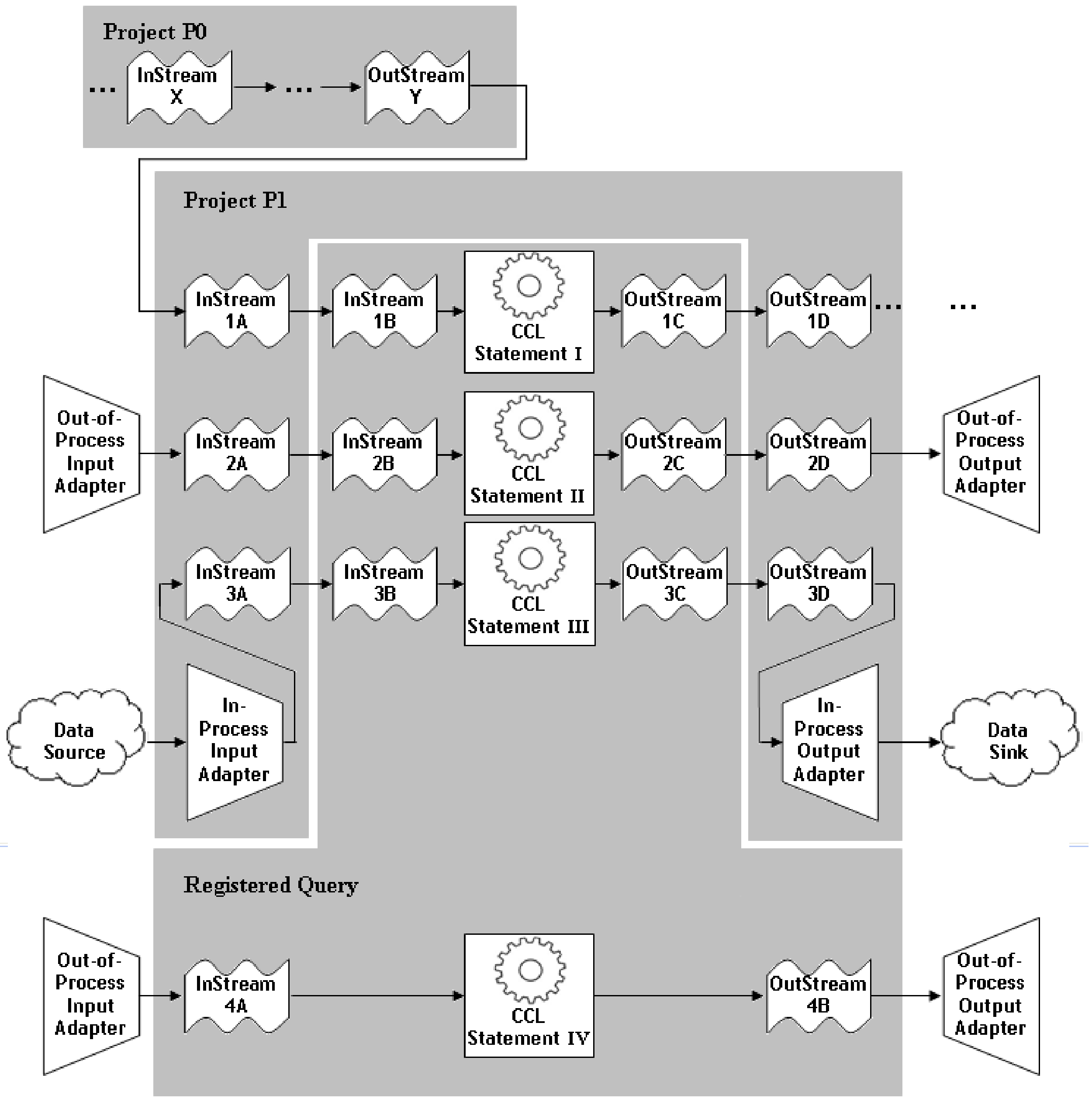


This example illustrates most of the key features and configuration elements of dynamic query registration. As you can see, we have created a project P1 that contains two streams. In this particular example, one of these streams is fed from an out-of-process input adapter, and the other stream writes 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 (no CCL statements).

It is important to understand that OutStream B is actually declared as an input stream when it is created. (This is explained in more detail later.)

Below 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 0, 1, 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).

Note that 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 of course may be referenced by CCL statements in the registered query.

To use dynamically registered queries, do the following:

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 diagram above. When you create the streams in this project, make sure to create all of them as "input" streams. (We'll explain this in more detail later.)
2. If some of these streams need to have input and output adapters attached to them, then attach those adapters.
3. If some of those streams will get their data from (or write their data to) another project(s) (e.g. P0 in the diagram), then create those other project(s).
4. Start execution of all of the projects, including P1 of course. 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 so that the binding operation can 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, e.g. P0) will read from or write to out-of-process adapters, then 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). It's not as obvious why streams OutStream1, OutStream2, and OutStream3 are created as input streams rather than output streams. The reason is that they, just like InStream1, receive data from outside the project that they are in. For example, output stream 2 gets input from the stream OutStream D (attached to CCL2), which is not a part of project P1. Thus, although OutStream2 is attached to an output adapter and writes to that adapter, OutStream2 must be categorized as an input stream. OutStream2 is BOTH an input stream and an output stream as far as project P1 is concerned. 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.

Client-side vs. Server-side Compilation

Compilation of a CCL project may be done on either the server machine or the client machine. Each method has advantages and disadvantages. Most of the differences are related to either resource consumption or security.

In general, client-side compilation is preferred if the server is heavily loaded and if registering a query is done frequently (which adds to the server's workload). In general, 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. Thus, for example, 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 (e.g. CPU, memory).
	Allows light-weight clients that do not need to have the Coral8 Engine (compiler) installed. This saves disk

	space and administrative work on the client, and also prevents client machines from having different versions of the Coral8 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 Coral8 SDK.

Guidelines

Below are some 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 will bind to.
- In the project that contains the streams that the dynamically registered query will bind to, all streams that will be bound to by the dynamically-registered queries must be created as "input", even if the dynamic query will write to them rather than read from them.
- In the project that contains the streams that the dynamically registered query will bind to, there should be only streams, no CCL statements.
- When you register a dynamic query, you must specify a name for each query (i.e. each set of CCL statements), and the name must be unique among the dynamically loaded queries in this workspace. (Without this name, you would 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, then the query will not be able to do anything useful.)

Binding a Registered Query's Stream to an Existing Stream

Any CCL statement may be in the registered query. The key step is to connect the input and output streams of the registered query to the streams in the already-existing project. In Coral8 Studio this can be accomplished with the binding mechanism. Note that binding streams in a query is optional.

The Coral8 SDKs provide an easy way to compile and register a query in one step -- the registerQuery() method. When you register the query, you will provide information that binds (associates) the stream name (the name used in the CCL statement) with the URI of the corresponding stream. Naturally, 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 your query. You may either bind those streams to existing streams in an existing project, or you may 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; instead, their 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:

1. Local streams are populated via the CCL statements in the dynamic query;
2. Input and output streams may be populated either:
    A. via the streams to which they are bound, or
    B. 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 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 Coral8 SDK that supports query registration also contains a "stopQuery()" function or method. (The exact name of the function or method depends upon 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

This section provides some troubleshooting tips that are specific to the Coral8 Perl SDK.

Additional troubleshooting tips are in Troubleshooting.

You get an error similar to the following:

```
Can't locate C8/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/c8_publish.pl line 10. . . .
```

The most likely cause is that you have not set your perl "include" path to include the Coral8 directory that includes the Coral8 perl modules. To configure your environment properly, see Prerequisites.

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 (Comprehensive Perl Archive Network modules). For a list of these modules, see Specifying the Path to the Library Files. If you have not already acquired these modules, you may need to acquire them. If you have acquired them, you need to set your perl "include" path to include these modules.

Monitoring Servers and Projects

This section describes how you can monitor servers and projects (query modules) that are running on those servers.

Status Information

The Coral8 Server provides extensive status information to help you monitor servers and projects. This includes information such as CPU utilization by a particular server, the number of messages received (processed) by a particular CCL query, etc.

Coral8 provides status information about the following types of entities:

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

Other types of status messages also exist, but we will focus on these since they are of the most use to typical customers.

All status information is collected and made available by the manager; you will not be getting status information directly from individual containers. This means that 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 different ways:

- Status streams. These are just like other streams in Coral8, but these carry status information that you can monitor and react to.

- Through the function/method calls provided in particular Coral8 SDKs, such as the C/C++ SDK and Java SDK.
- Server plugins

Each of these is described in more detail below.

Status Streams

Each manager provides one status stream, which may contain messages about each of the different types of entities (Manager, Container, workspace, project, etc.) for which Coral8 provides status info. Some types of status information, such as the number of messages a particular project has sent, are published at regular intervals. (In most cases, the default rate is once per second, but you can configure the rate.) 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 occured. 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 Coral8 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", i.e. 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_AdapterReadPallet/Out`
MessageGroup	The combination of MessageGroup and MessageName specify the "topic"

and MessageName	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*.
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 (i.e. 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*.

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 Coral8 SDKs, including C/C++, Java, and .NET, allow you to get status information by calling particular functions/methods.

To get information about a particular "object" (e.g. a container or a query), you must specify two pieces of information:

- the unique ID of the object,
- an indicator of which information you want about that object.

To specify the ID of the object, you use the objectID. The objectID was described briefly above. Below we show how to compose the object ID for each type of object:

Object Type	**Object ID ("Path")**
Manager	Manager URI
Container	Container URI
Workspace	Workspace name
Project	The ObjectID is the full path to the given project in the form: <WorkspaceName>/<ProjectName> Where ProjectName is the project's load name, i.e. the name that is displayed in Coral8 Studio's Explorer View when you load the project (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: <WorkspaceName>/<FullCclPath>/<StatementNumber> Where <FullCclPath> is the "path" of the loaded module as you see it in Studio's Explorer View (e.g. `Project1/SubModule1/SubModule2`) and <StatementNumber> is the sequential number of the CCL statement in the module.

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

In general, the Message Name indicates the specific information that you want, e.g. the percentage of CPU time utilized by the object (such as a container). Because some types of objects have similar information (e.g. 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 uniquely indicate which information you want. Thus, to get a specific value (e.g. the number of input messages received by a particular query), you must specify all three of the pieces of information necessary to uniquely identify the message; the ObjectID, MessageGroup, and MessageName.

First you'll get a status "object" that contains information about a Manager, a Workspace, or a Project (a "top module"). Once you have that object, you will 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 status objects for top query modules ("projects") have 10 different types of messages (MessageNames), then when you retrieve a status object for a particular query you will be able to extract 10 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

NOTE: This section applies only to the Enterprise Edition of Coral8 Engine. If you are using the Professional Edition, you may skip this section.

The Enterprise Edition of Coral8 supports a user authentication feature. This allows an administrator to limit access to the Coral8 Server, or to certain objects within the server (such as streams, workspaces, etc.). 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 passing a user name and password when calling certain API functions.

The following operations require authentication:

- Connecting to a stream (as a subscribing or publisher)
- Getting a stream schema
- Resolving a URI
- Starting or stopping a Coral8 project
- Getting the status of a server
- Getting the status of a workspace
- Getting the status of a project (also called an "application")

The Coral8 C/C++, Java, and .NET SDKs support user authentication.

In general, you will follow these steps:

1. Get an empty C8UserCredentials object.
2. Fill in the C8UserCredentials object with the user name and password.

3. Do the desired operations (e.g. connect to a stream)
4. Destroy the credentials

(This assumes that you have already configured the server to turn on the user authentication feature.)

Note that, depending upon which user authentication rules have been set up, 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 Coral8 Administrator's Guide.

If you will be creating your own plugin to do custom authentication, see How to Implement Manager HA with the Coral8 Generic Plugin.

User-Defined Functions and Plugins

One of the most powerful features of the Coral8 Engine is the ability to extend the server by writing User-Defined Functions (UDFs) and server plugins.

This chapter provides a conceptual overview of UDFs. A detailed explanation of how to write and use UDFs in the C/C++ language is in User-Defined Functions.

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 Coral8 Server, you may also write user-defined functions by using CCL's CREATE FUNCTION statement. See the CCL Reference Guide 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*.

User-Defined Functions

You create a User-Defined Function (UDF) by writing your own code (in the C/C++ programming language), dynamically linking that code into the server, and then calling that code from within a CCL statement.

For example, suppose that you want 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 Coral8 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, 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 may appear in most parts of a CCL statement where a general expression may appear.

A UDF may 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;
```

For each row, if the same UDF is called with the same parameter value(s), the server may 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 result for use some of the times that 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 User-Defined Function can contain almost any code that you can write in any of the languages for which Coral8 provides an SDK that supports UDFs.

UDFs allow users to provide for their own specialized needs. The range of capability that you may add is almost unlimited. The UDF may even go "outside" the environment of the Coral8 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).

UDFs may be scalar functions or aggregate functions. Scalar functions return a value based on a single set of inputs (e.g. 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.

Requirements

Since the server must be able to find the library that contains your User-Defined Function, load the library into memory, call your UDF, and pass 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 User-Defined Function(s) 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.)

- 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 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 (also called a Dynamic Link Library) for the 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 Coral8 SDKs are enhanced to support UDFs.

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

- A C-language file that contains your User-Defined Function 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

Users may write aggregate functions.

For a user to write an aggregate function, the user must have a place to store her data in 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 were to 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 (i.e. stack-based variables) can't be used because, of course, the value may be written over in between calls to the function.

Thus the user's aggregate function must work with the server to store data in a place that will persist after the UDF returns from its call. Coral8 provides a pair of functions that allow 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" (e.g. 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 from the window. If you are writing a SUM() function, for example, and you only want the sum of the 3 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 was called with the newly-added value, then C8PositiveMessage() will return C8_TRUE, and of course 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

Coral8 software requires the description of the User-Defined Function to be in an XML file that is put in the "plugins" directory of both the server and Studio. The file name should have the extension ".udf" so that the server and Studio will 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 convenience allows grouping of related functions and/or library modules. As expected, 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 user function. For UNIX-like operating systems, this will be a .so library. Note that on UNIX-like operating systems, you should not preface the library name 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 (e.g. "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. These names are passed to the user for similar uses.

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 (e.g. "C8Float") rather than the underlying C types (e.g. "double"). The Coral8 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.

Coral8 SDKs that Support UDFs

Currently, the Coral8 SDKs that support UDFs are:

- C/C++ *Coral8 C/C++ SDK*

For much more detail, see User-Defined Functions.

Plug-Ins

Like User-Defined Functions (UDFs), plugins are executable code (typically written by users or third parties) that expand the functionality of the server (like the user authentication plugin (see The User Authentication Plugins)) or implement customization of existing functionality (like the server status plugin, which can be used to help implement High Availability (see How to Implement Manager HA with the Coral8 Generic Plugin)).

Unlike UDFs, plugins are called by the server, rather than called from the user's CCL statements.

Every plugin must have at least three functions:

- an "initialize" function
- an "execute" function
- a "shutdown" function

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

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

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

Naturally, 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 (coral8-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 (High Availability) 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 coral8-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

Coral8 SDKs that Support Plugins

Currently, the Coral8 SDKs that support plugins are:

- C/C++ *Coral8 C/C++ SDK*

For much more detail, see User-Defined Functions.

Other SDKs may support plugins in the future.

Remote Procedure Calls, Database Queries, and Public Windows

As you know, Coral8 Server can get data from input adapters. Coral8 Server can also get data from, or invoke processing on, remote servers, where the remote servers may be database servers (e.g. MySQL, SQLServer, etc.) or Remote Procedure Call (RPC) servers.

Queries can also go in the opposite direction -- you can write a program that will read data from a *public window* (a window that exists inside Coral8 Server but can be accessed from outside Coral8 Server as though the window were a database table).

This chapter provides an overview of how to integrate your Coral8 Server with remote servers.

Please note that this chapter does not completely document this topic. You should also read:

- Coral8 Administrator's Guide - This explains how to configure your **coral8-services.xml** file to tell your Coral8 Server how to connect to the remote servers.
- Coral8 Reference Guide - This documents the CCL language statements (including the database statement ("EXECUTE STATEMENT DATABASE") and EXECUTE REMOTE PROCEDURE) and clauses (e.g. database subquery) that are capable of accessing remote servers. The CCL Reference also documents the CREATE PUBLIC WINDOW statement, which is used to create a public window that can be accessed externally, and Coral8's SQL, which can be used to query a public window.

Remote Procedures

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

Overview: Calling a Remote Procedure from a CCL Statement

Coral8's Remote Procedure Call (RPC) feature allows CCL statements to execute a function or procedure that is running outside the Coral8 Server process, either on the same computer as Coral8 Server, or on a different computer. (Note: We use the term "RPC" refer to both "procedures" (which do not return a value) and "functions" (which return 1 or more values).)

How to Choose Whether to Use an RPC or a UDF

Reasons for using an RPC:

- The RPC code exists in a form that cannot be made into a .so or .dll file that Coral8 Server file can use. Reasons for this might include:

    - The code is supplied by a 3rd party and is supplied only as an executable library, not source code.
    - The code is not written in a language that Coral8 SDKs support for UDFs.
- You want the RPC to be in a separate process so that if the RPC crashes it won't bring down Coral8 Server.
- The RPC is resource intensive and you want to run it on a separate machine from Coral8 Server so that you spread the workload.

Reasons for using a UDF:

- UDF calls are usually much faster than RPC calls. (This is true for both CCL UDFs and C/C++ UDFs.)
- If the UDF can be written in CCL, it is often much easier to write the UDF in CCL than to write an RPC and configure the server to call that RPC.

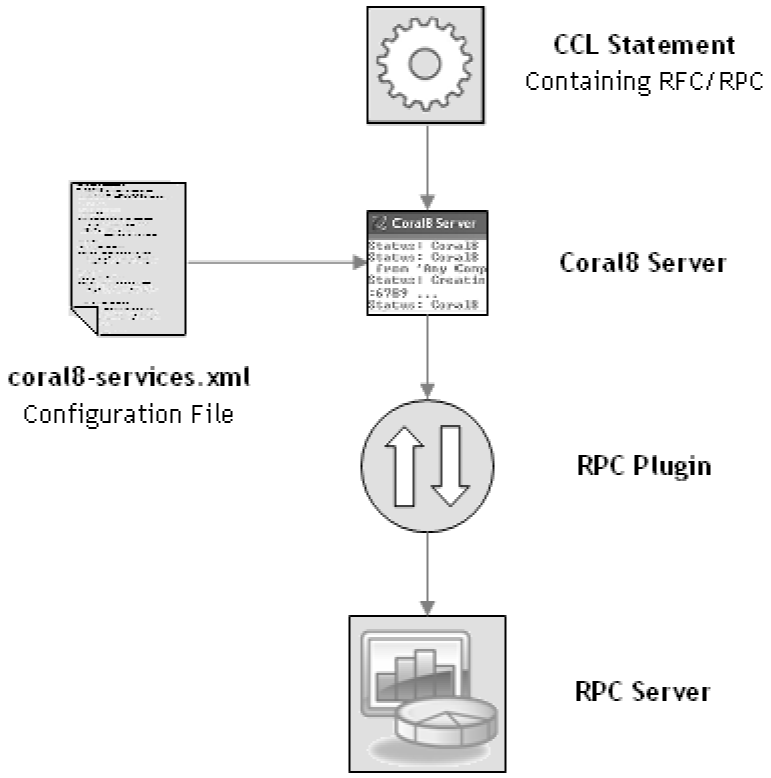
What Is an RPC?

Remote Procedure Calls (RPCs) are somewhat like User-Defined Functions (UDFs). Both UDFs and RPCs allow you to extend Coral8 Server by writing custom code. The difference between an RPC and a UDF is that a UDF runs as part of the server process. An RPC, on the other hand, runs as part of a separate process.

In many cases, you could accomplish the same goal with either a UDF or an RPC. The most common reasons for choosing one over the other are listed below:

The Components

From the perspective of the CCL statement, an RPC is a simple operation. However, "underneath the covers", each RPC call involves multiple steps. The diagram below shows the components involved and how they are related.



Coral8 RPC functionality requires the following software components:

- A Coral8 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 Coral8 CCL Reference.
- A Coral8 Server
- A section of the coral8-services.xml file that describes the RPC plugin and allows the Coral8 Server to call it. (RPC Plugins are explained later.) Each RPC plugin must be described in a separate section of the coral8-services.xml file.

  For each remotely callable procedure, Coral8 Server must know the name of the library file that contains the corresponding RPC plugin.
- An RPC plugin that interfaces between Coral8 Server 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 Coral8 Server.

  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. Coral8 Server itself does not need to know how to communicate with every remote procedure that might be written; the plugin handles the communications.

The RPC plugin must be compiled as a dynamically linkable library (.dll file) or shared object library (.so file) so that Coral8 Server can call it.

- An external executable program that includes a procedure or function that can be called 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 Coral8 Server 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 coral8-services.xml file) and may also pass values to the plugin, which are then passed on to the RPC server.
2. The Coral8 Server uses the service name and the information defined for the service in the coral8-services.xml file to determine which RPC plugin to use and which function(s) in that plugin to call. The Coral8 Server may also read some plugin configuration parameters (also defined in the coral8-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.

More details about the individual components are below.

The 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 we show 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 that a new row arrives in any of the streams involved in the CCL statement.

Note that, as with database subqueries, Coral8 Server has no knowledge of when information on the remote system has changed. The RPC cannot notify Coral8 Server that data has changed; the server will 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 Coral8 Reference Guide.

The coral8-services.xml File

The coral8-services.xml file must contain a section for every remote service, before the service can be used 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 will depend on the RPC plugin and remote procedure you are using. See the Coral8 Administrator's Guide or *HTTP and SOAP Plugin Preferences* for more details.

The Coral8 Server

Coral8 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 Coral8 Administrator's Guide.

The RPC Plugin

RPC plugins are compiled .dll files (on Microsoft Windows) or .so files (on UNIX-like operating systems). These libraries can be called by the Coral8 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 that 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.

Coral8 provides two generic plugins that allow you to make remote procedure calls to RPC servers that handle SOAP (Simple Object Access Protocol) 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 coral8-services.xml file.

Note: Do not confuse the RPC plugins discussed in this chapter with Coral8 Server plugins discussed in HTTP and SOAP Plugin Preferences. 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 Coral8 Server.

The RPC server may run on the same computer as the Coral8 Server, or on a different computer.

The RPC server may "serve" many different "clients"; Coral8 Server, RPC plugin, and CCL query may be only one of many clients for the RPC server. Furthermore, the "client" (Coral8 Server, plugin(s), and CCL query(s)) may be clients of many different RPC servers. One RPC server might check email for viruses, while another looks up historical stock information, and yet another looks up customer information.

The RPC server may be written by the customer or by a 3rd party.

The RPC may be written in any language for which you can find or write software that allows the RPC to respond to remote procedure call requests. You are not limited to the languages for which Coral8 provides SDKs.

Additional Information about Remote Procedure Calls

Coral8 Server calls a remote procedure only when:

- a new row arrives in one of the input streams referenced by the CCL statement that contains the remote procedure call AND
- the information in the cache has expired. (You may set a coral8-services.xml file parameter to determine how long the server will cache responses from the remote procedure calls.)

Note that RPC calls are not "blocking calls" within Coral8 Server. See Remote Requests, Synchronization, and Performance for more details.

Generic HTTP and SOAP Plugins

The Coral8 product includes two example RPC plugins:

- The HTTP plugin may be used to send HTTP POST requests. The source code for this plugin is in the file `c8_rpc_http.cpp`.

- The SOAP plugin may be used to send SOAP requests. The source code for this plugin is in the file `c8_rpc_soap.cpp`.

If you have remote procedures that can be called via the HTTP or SOAP protocols, then you may use these RPC plugins instead of writing your own plugin. (Note that when you specify the name of the compiled library file in the coral8-services.xml file, you should specify only the "base name" c8_rpc_http_soap_lib. The server will automatically append 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" will also be prepended, if necessary.)

The Coral8 HTTP plugin allows you to pass 0 or more parameters and get back 0 or 1 parameters.

The Coral8 SOAP plugin allows you to pass 0 or more parameters and get back 0 or more parameters.

If you need to retrieve more than 1 parameter, you should use the SOAP plugin or a custom plugin.

Source code for both libraries is in the `server\sdk\c\examples` subdirectory of your Coral8 installation directory. Typically, this is either:

```
C:\Program Files\Coral8\Server\sdk\c\examples
/home/<userID>/coral8/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 (http://www.w3.org/TR/soap12-part1/ and http://www.w3.org/TR/soap12-part2). The SOAP RPC plugin expects any number of input columns from the CCL statement and can produce any number of output columns. For information about the conversion of CCL datatypes into XSI types and vice versa, please see Appendix I of the Coral8 CCL Reference.

The SOAP plugin recognizes SOAP errors and returns nothing to the CCL statement if it encounters such an error.

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

Since Coral8 provides these two plugins in source code form, you will need to compile them before using them, and you will need to put the compiled code into the server's "bin" directory.

RPC Plugin for CSV Files

Coral8 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 **coral8-services.xml**, located in the **conf** subdirectory of your Coral8 Server installation directory. The file itself contains comments and an example configuration, and the *Coral8 Administrator's Guide* provides instructions for enabling remote procedure calls.

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

- **BaseFolder**: When you call this plugin and specify a file, the file must be located somewhere beneath the base folder specified here. If you don't specify a value for this parameter, Coral8 Engine uses the value of the **ReadWriteBaseFolder** preference specified in **coral8-server.conf** instead. Optional.
- **Format**: The format of the file, in the form "**TitleRow**= true | false, **TimestampColumn**= true | false." The defaults are true and false, respectively. Optional.

Using the Plugin

When you use the plugin in a query, you provide two parameters:

- **Filename**: 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. Required.
- **Format**: The format of the file, in the form "**TitleRow**= true | false, **TimestampColumn**= true | false." The defaults are true and false, respectively. Optional.

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 Coral8 RPC Plugin

You may write your own plugin to communicate with remotely-callable procedures. You may need to do this if you want to call a remote procedure that does not use SOAP or HTTP protocols.

Writing your own Coral8 RPC Plugin is non-trivial, and is not fully documented in this manual. A general overview is below. You may also get some information by looking at the SOAP and HTTP Plugins that Coral8 provides. On Microsoft Windows, these are typically located in:

- `C:\Program Files\Coral8\Server\sdk\c\examples\c8_rpc_http.cpp`
- `C:\Program Files\Coral8\Server\sdk\c\examples\c8_rpc_soap.cpp`

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

- `/home/<userid>/coral8/server/sdk/c/examples/c8_rpc_http.cpp`
- `/home/<userid>/coral8/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 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 may simply return immediately to the caller.
- An execute function The execute 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 The shutdown function is called exactly once when the RPC server shuts down. Typically, a shutdown function closes connections and releases resources. If no particular shutdown work is required, the shutdown function may simply return immediately to the caller.

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

For simplicity, it is recommended that a separate .dll or .so file be used for each remote service, but this is not required. 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 may each use different initialize, execute and shutdown functions, or may share one or more of these functions.

If you are going to write your own plugin, you will need to use a Coral8 SDK. The Coral8 SDKs that support RPC Plugins are:

- C/C++ *Coral8 C/C++ SDK*

For more detail, see RPC Plugins.

Compiling a Coral8 RPC Plugin

The process for compiling a Coral8 RPC Plugin is much the same whether you've written your own or are using one of Coral8's.

For more information, see RPC Plugins.

Remote Database Queries

This section documents remote database queries.

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

We use the term "remote database server" to refer to a database server that runs outside the Coral8 Server process, whether on the same computer as Coral8 Server or on a different computer. Coral8's remote database query feature allows CCL statements to execute a query on a remote database server. The query may be a SELECT statement that returns data, or may be some other type of query/statement, such as an INSERT, UPDATE, or DELETE.

A query module may also use a remote database query to query a public window in another module. For more detail, see Public Windows.

The CCL statement may use parameters to pass information from CCL statements to SQL statements. For example, if the current row in the query on Coral8 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 Coral8 Server that the SQL query text should be updated to look like `"... WHERE DEPT = 'Accounting'"`. For more detail see the Coral8 CCL Reference Guide.

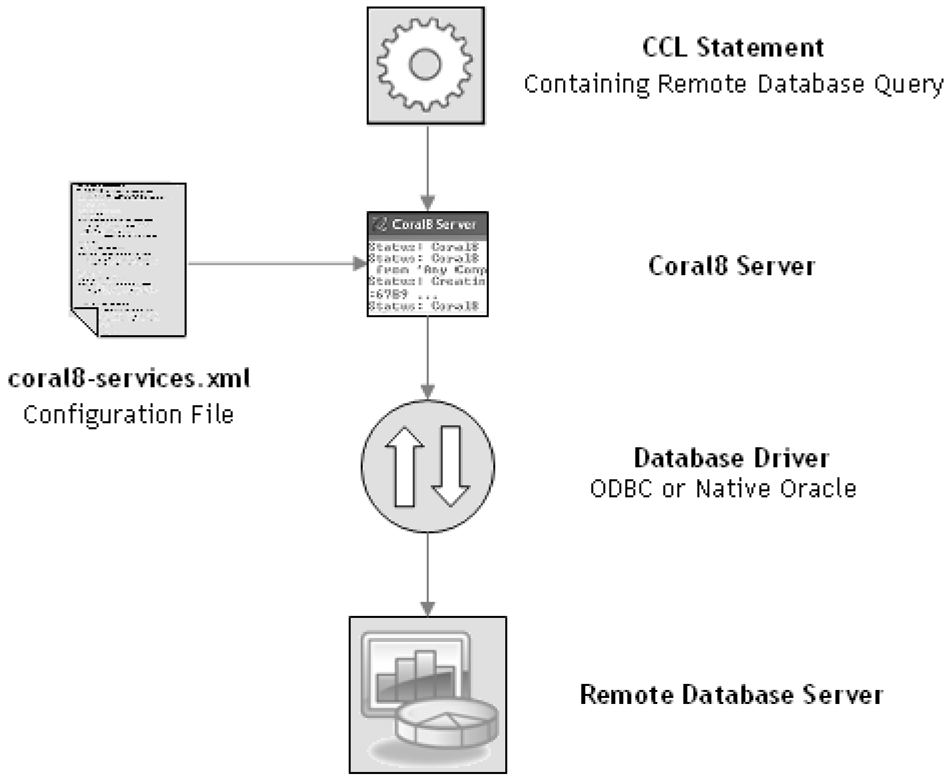
What Is a Remote Database Query?

We use the term "remote database query" to refer to either of the following:

- a database subquery, i.e. a subquery that selects data from a remote database server
- a Database Statement (EXECUTE STATEMENT DATABASE ...)

The Components

From the perspective of the CCL statement, a remote database query is a simple operation. However, "underneath the covers", each query involves multiple steps. The diagram below shows the components involved and how they are related.



Coral8 remote database query functionality requires the following software components:

- A Coral8 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, please refer to the Coral8 CCL Reference.

- A Coral8 Server
- A section of the coral8-services.xml file that describes the remote database server as a "service" and allows the Coral8 Server to call it. Each remote database server must be described in a separate section of the coral8-services.xml file.
- A database driver (such as an ODBC driver or some other appropriate driver) that interfaces between Coral8 Server and the remote database server.
- 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 coral8-services.xml file) and passing the SQL statement.
2. The Coral8 Server uses the service name and the information defined for the service in the coral8-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 Coral8 Server, which then makes the values available to the CCL query.

More details about the individual components are below.

The CCL Statement

The CCL statement may be either a DATABASE statement or may be a subquery that makes a remote query call. Below we show 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 that a new row arrives in any of the streams involved in the CCL statement.

Note that, as with database subqueries, Coral8 Server has no knowledge of when information on the remote system has changed. The remote database server cannot notify Coral8 Server that data has changed; the server will not see updated information until a row arrives in one of the query's streams and the server then executes a statement on the remote database server. There are indirect ways that the remote database server can notify Coral8 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 a Coral8 Server.

For more information about the syntax of these CCL statements and clauses, see the Coral8 Reference Guide.

The coral8-services.xml File

The coral8-services.xml file must contain a section for every remote service, before the service can be used 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 will depend on the remote database server you are using. See the Coral8 Administrator's Guide for more details.

Coral8 Server

Coral8 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 Coral8 Administrator's Guide.

The Driver

The driver that you use will be based upon the database server that you want to access. The driver must be designed and configured to work with the specific remote database server (e.g. MySQL, etc.) that you want to use. The driver must be able to contact the database server, pass queries to it, etc.

The Remote Database Server

The remote database server runs as an independent process, not as part of Coral8 Server.

The database server may run on the same computer as the Coral8 Server, or on a different computer.

The database server may "serve" many different "clients" -- Coral8 Server, and CCL query may be only one of many clients for the remote database server. Furthermore, the "client" (Coral8 Server, plugin(s), and CCL query(s)) may be a client of many different database servers. For example, one database server might look up historical stock information, while another might look up current customer information.

Additional Information

Coral8 Server only calls a remote database server when a new row arrives in one of the input streams referenced by the CCL statement that contains the remote database query AND the information in the cache has expired. (You may set a coral8-services.xml file parameter to determine how long the server will cache responses.)

Database subqueries are not "blocking calls" within Coral8 Server. See Remote Requests, Synchronization, and Performance for more details.

The Coral8 Reference Guide has information about the syntax of database subqueries and the EXECUTE STATEMENT DATABASE CCL statement.

Information about the coral8-services.xml file is in the Coral8 Administrator's Guide.

Reading and Writing BLOBs on External Database Servers

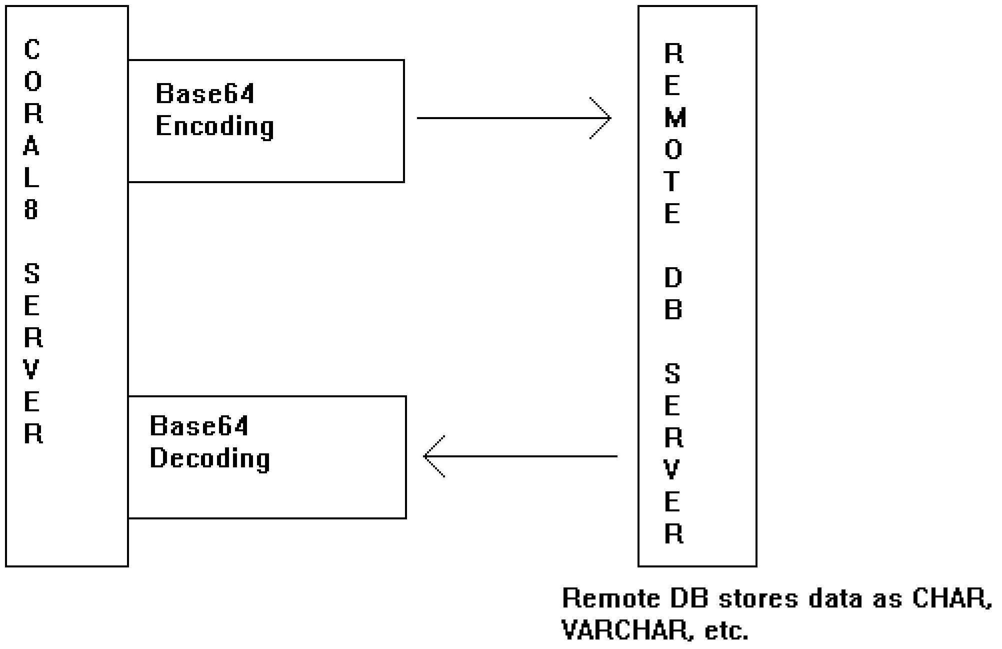
This section covers the following situations:

- You want to read data from or write data to an external database server that supports the BLOB data type.
- You have BLOB data on Coral8 Server and want to write it 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.

Background

Many database servers support character data but not BLOB data. BLOB data can be stored on such servers by converting it to a character string format before sending it. Naturally, when reading data back from such servers, the data must be converted from the character string format back to the original Coral8 BLOB format.

Coral8 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's character data type has a limit of N bytes per character string, then only 3/4 * N bytes of BLOB data can be stored. For example, if the external database server has a limit of 2000 bytes per character string, then the longest BLOB it can store will be 1500 bytes, which will of course have expanded to 2000 bytes after base64 encoding. (Note: 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.)

Keep in mind also that the DBDriverCharBufferSize limits the maximum size of data that you can write to or read from remote database servers.

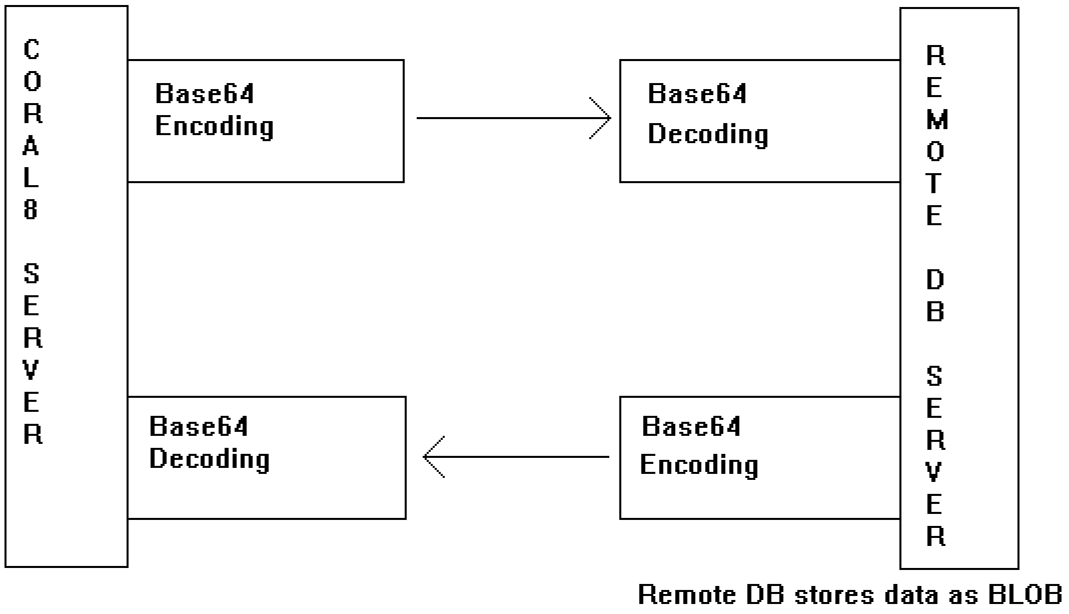
Reading From and Writing to External Database Servers that Do Not Support BLOBs

Coral8 Server takes care of doing the base64 encoding and decoding for you if BLOB data is to be stored as character data. You do not need to do anything except 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's character data type (e.g. VARCHAR) can hold, or may be limited by DBDriverCharBufferSize. (Remember that 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 Coral8 Server can exchange with the remote database server.)

Reading From and Writing to Database Servers that Support BLOBs

Since Coral8 Server sends and receives the data in base64 format, if you want the data to be stored in its original form on the external server, then you'll need to explicitly decode the data on the external server before storing the data on the external server, and you'll need to explicitly encode the data on the external server after reading the data from the external server.

If the external database server does support BLOB data, you might think that reading and writing BLOB data between the external server and Coral8 Server would be easy and you would not need to take into account that the data takes 4/3 as much space when converted to base64 format. Unfortunately, however, the current implementation of the Coral8 Server does not provide transparent reading and writing of BLOB information on external database servers. When writing to an external database server, Coral8 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, Coral8 Server automatically decodes data EVEN IF THE EXTERNAL SERVER SUPPORTS THE BLOB DATA TYPE. If all you want to do is store the data on the external server and then read it back to Coral8 Server, this is not a problem (although it is somewhat inefficient). However, if you want to process the data on the external server, then you need to store the data as a proper BLOB on the external server. For example, if you have an image that you want to process on the external server, then you need to store that image in its original (BLOB) form so that the image-processing software can interpret it correctly. If you left the data in base64 format, the image-processing software on the external server probably wouldn't know how to interpret and process the data.



For example, if you write a database subquery to read data from a BLOB column on an external database server, your query will look something like the following:

```
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;
```

Note specifically that the external database server is told to encode the value in the photo_blob column as a base64 value; this is because Coral8 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(), of course. If the database server does not provide such a function, you may need to write one yourself.)

Similarly, to write data to a BLOB column on an external database server, we must tell the external database server to decode the data from base64 into the external server's native BLOB format, since Coral8 Server will have encoded the data in base64.

```
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 does, of course, consume some resources. It also means that if the input or output parameter of the encoding/decoding subroutine has a maximum size limit (typically 2GB), then you'll still be limited to 3/4 of that size. Note that if you only want to read and write the data on the external database server to store it, and you will never process it as a BLOB on the external database server, then you can simply store it as character data and avoid the extra processing of encoding/decoding required if you store it as BLOB data on the external server.

When reading BLOBs from external databases, the maximum size of the BLOB being read in by Coral8 is limited by the limits of the database driver (e.g. an ODBC driver). When writing BLOBs to external databases, the size of the BLOB being written to the database is limited by Coral8's DBDriverCharBufferSize setting (in the file named `coral8-services.xml`) which by default is only 4k. See the Coral8 Administrator's Guide for information about this configuration setting.

When reading BLOB data from an external database, you must transform the BLOB to a base64 encoded string before Coral8 gets the data. Similarly, when writing BLOB data to an external database server, you must transform the data from base64 format (which is what Coral8 sends) to BLOB format, if you want the external database server to store the data in BLOB format.

Different database servers will have different packages or functions that you must use to create a base64 encoded string.

When reading BLOBs from external databases the maximum size of the BLOB being read in by Coral8 is limited by the limits of the database driver (e.g. an ODBC driver). When writing BLOBs to external databases, the size of the BLOB being written to the database is limited by Coral8's DBDriverCharBufferSize setting (in the file named `coral8-services.xml`) which by default is only 4k. See the Coral8 Administrator's Guide for information about this configuration setting.

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, depending upon factors such as network bandwidth and latency, the load on the remote service, etc. The delay in getting data back from the remote call may delay the output of rows from the queries.

Coral8 Server uses multiple techniques to minimize the delay. The techniques include:

- caching data from the remote service
- internal parallelization

Although a complete explanation of these features is beyond the scope of this document, we provide some basic information below. Some additional information is in the Administrator's Guide and some of the Coral8 Technical Notes.

Caching

To reduce the delay, the server may be configured to cache data that has been retrieved from a remote server and use the cached values rather than re-issue the query. To ensure that the cache is refreshed, customers may set a maximum limit on the length of time that data may stay in the cache. The user may also set other caching-related parameters.

See the *Coral8 Administrator's Guide* for more information about caching.

Internal Parallelization

As noted above, although CCL semantics specify that rows be kept in order and processed based on their row timestamps, that does not mean that Coral8 Server cannot use concurrency. 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. Coral8 Server might 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 Coral8 Server's cache already contains the results of a previous request for beta data. In this case, Coral8 Server simply stores the "beta" remote query results until it has received the "alpha" remote query results and processed the rows that needed that data. The server then moves on to process the later rows (those that issued the "beta" request).

Coral8 Server ensures that an input row is processed in order irrespective 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 will be 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 will not re-order the rows.

Public Windows

Coral8 Server's public window feature allows the rows in a window to be queried by any of the following:

- a separate project running on a Coral8 Server, which can query the public window using a database subquery or the CCL database statement (see "Database Statement" and "Database Subquery" in the *Coral8 CCL Reference* for more information)
- an application written using one of the Coral8 SDKs
- through the Coral8-provided ODBC driver for public windows

For more information about public windows, see "Public Windows" in the *Coral8 CCL Reference*.

Coral8 Project

If you will access the public window from another Coral8 project, then you must configure the Coral8 Server doing the query so that this Coral8 Server can access the public window as though it were a table in a remote database -- even if the project with the query and the project with the public window are on the same Coral8 Server. Specifically, you must create a database service entry in the coral8-services.xml file; this will make the public windows in the project look like a "service" that the Database Statement or a database subquery can access. For information about the `coral8-services.xml` file and its PublicWindow entries, see the *Coral8 Administrator's Guide*.

The following Coral8 SDKs allow you to write a program to read from a public window:

- Coral8 C/C++ SDK. See Querying a Public Window.
- Coral8 Java SDK See Querying a Public Window.
- Coral8 .NET SDK See Querying a Public Window.

Coral8 SDK

To query a public window using an SDK, you call an SDK function and supply a query written using Coral8 SQL syntax. (See "Coral8 SQL" in the *Coral8 CCL Reference* for information about the syntax.) The SQL statement must be a SELECT statement, and you must submit only 1 statement per query. The function will return a result set containing zero or more rows. You may then access each individual row in the result set.

You may want to test the Coral8 SQL query from inside Coral8 Studio before using it in an SDK call. Use the Query Public Windows in Module command on the Debug menu.

Coral8 Public Windows ODBC Driver

Coral8 Server includes an ODBC driver for use with public windows. When you set up a new User DSN for the Coral8 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:

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

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

The Coral8 Public Windows ODBC driver provides the core level of ODBC compliance, as defined by Microsoft (http://msdn.microsoft.com/en-us/library/ms714086(VS.85).aspx).

As you design a system that uses public windows, remember the following: Storing and updating public windows requires Coral8 Server to consume both memory and CPU cycles, of course, so you may see a reduction in performance of other queries when you add public windows or add queries of public windows.

Engine Control: Command-line Tools

This chapter describes how to do the following tasks with command-line tools.

In order to deploy a Coral8 application, you must be able to do the following things:

- Start Coral8 Server
- Access a running Coral8 Server
- Create a query module and its associated schema files
- Compile the query module using the Coral8 Compiler
- Create a workspace on the Server
- Execute the query module on the Server
- Stop the query module on the Server
- Stop the 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 (e.g. 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 Coral8 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, then c8_client will prompt you for a password.

Start the Server

The normal method for starting Coral8 Server is documented in the Coral8 Administration Guide. It may be started the same way regardless of whether Coral8 applications will be deployed from the Studio or from one of the SOAP interfaces. The Server may be started from the command line in both Microsoft Windows and UNIX-like environments. The command to invoke the Server from the command line is:

```
c8_server [options]
```

The following parameters and options can be used with the server.

--help	Prints a list of command line parameters
--version	Prints the current version number
--service=install	Microsoft Windows Only: This option installs Coral8 Server as a Microsoft Windows service.
--service=uninstall	Microsoft Windows Only: This option uninstalls Coral8 Server as a Microsoft Windows service.
--service=start	Microsoft Windows Only: This option starts the server when it is installed as a service.
--service=stop	Microsoft Windows Only: This option stops the server when it is installed as a service and if it is running.
--config=*filename*	Sets the server configuration file to the specified file name.
--clean-state	Starts the server in a "clean state". All modules that were running or loaded the last time that the server was shut down will be cleared. If any modules were using the "persistence" feature, all persisted data will be cleared.

Start the Server on UNIX-like Operating Systems

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

```
coral8-server.rc start
```

The file coral8-server.rc is a shell script that sets the environment appropriately and then starts the server.

If you try to start the server directly (e.g. 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 (e.g. the LD_LIBRARY_PATH environment variable) is not set properly.

Start the Server on Microsoft Windows

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

```
C:\Program Files\Coral8\Server\bin\c8_server.exe -c=C:\Program
Files\Coral8\Server\conf\coral8-server.conf
```

(This command should all be on one line, of course, even if it's not shown on one line in this manual.)

The "-c=C:\Program Files\Coral8\Server\conf\coral8-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" (i.e. without any of the other command-line parameters), you will probably see a message that includes:

```
license file was not found
```

If you do not specify the configuration file, then the server will not know where to look for the license file.

Access a Running Coral8 Server

It is important to know the URI of your Coral8 Server. The URI is the only means of identifying which server a SOAP request should go to. Every command or API call that issues one or more SOAP requests to a server needs to know the URI of the Server. Some of the API environments may 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>
```

e.g.

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

If your server contains separate manager(s) and container(s), use the URI of the manager. Coral8 Container Servers are used to run queries, in-process adapters, and user-defined functions. Coral Manager Servers "manage" containers by assigning tasks to them and by monitoring their status.

If you are using a secure server (i.e. 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 Coral8 Server (or stream or ...) 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.

Coral8 Project Files

Coral8 projects are stored in pairs of files; by convention, these files have extensions of "`.ccp`" and "`.ccl`". These files contain not only CCL statements, but also an XML tag structure. Coral8 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 will need to know the XML tag structure. A query module file may be created and edited using any text editor or XML editor of the developer's choice. However, the developer must create the proper XML tags and content. An XML editor may assist in making sure that all tags are correctly structured, but using the tags in the right order and providing the proper content for each tag is not a trivial exercise. 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 Studio or RegisterQuery() to create your projects or register your queries.

The XML tag structure used in these files is not fully documented, and there are no plans to document it. 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). Although you may create your own .ccp and .ccl files directly (rather than through Studio or RegisterQuery()) if you wish, we do not recommend this.

Coral8 Schema Files

Coral8 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 Coral8 Studio's query editor.

Create a Workspace on the Server

In order to deploy a Coral8 query module, the server must have a workspace that the query module can be run inside. Workspaces are logical divisions within the 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 Server
workspace-name	The name of the workspace
workspace-description	The workspace description is a string that is purely a comment. This parameter is optional.

The workspace name must conform to Coral8 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 a project can be deployed to a Server, it must be translated from its "source" form (.ccp/.ccl files and .ccs files) to an internal "object code" form by the Coral8 Compiler. There are two ways that you can 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 will compile the program and start 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

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 `coral8/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>/coral8/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\Coral8\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. Note that 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 may 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 your own adapter, and that adapter needs to read schema information from a file with the C8SchemaReadFromFile(filename) function, you'll specify the name of the .ccx file that you created by compiling the schema file.

The Coral8 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 Coral8 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 Coral8 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 *progname*.

	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 this flag is 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 this flag is 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 Coral8 Studio. The default value is false.
--optimize:filter_asap=true | false	If this flag is 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 this flag is set to true, the compiler eliminates redundant primitives (more information is provided later in this page). If this flag is set to false, elimination of primitives is not done. The default is true. This flag is rarely used.
--persistence=true | false	Enables or disables persistence for the project. When enabled, Coral8 Server 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, Coral8

	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 Coral8 Studio on a computer other than the one on which you compile the project, the absolute path can cause an error when Coral8 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. Note that the compiler compares paths for imported files against the path to the specified repository without accessing the file system. Attempts to import files outside the repository generate an error message.
--restart-on-failure=true | false	If true, Coral8 Server 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 Coral8 Server when each message arrives (useservertimestamp). Defaults

	to inorder.
--synchronization:maximum-delay=*value*	Sets the maximum time, in microseconds, that Coral8 Server 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 Coral8 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 this flag is 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 this flag is set to true, then the compiler issues warnings if deprecated parts of the language are used. If this flag is set to false, no such warnings are issued. The default is true.
--warn:implicit_conversions=true | false	If this flag is set to true, then the compiler issues warnings if implicit data type conversions are used. If this flag is set 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 Coral8 Studio. 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*.

The Coral8 CCL compiler may 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 would like the output written to. Remember that 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 Coral8 Studio'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, 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 Coral8 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 will merge 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 would like 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.

Note that 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, the compiler reads .adl (Adapter Definition Language) files, which may be provided by Coral8 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* for details), then 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.

Execute the Project

Once the .ccx version of the query module has been compiled, it can be sent to the server and started 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, i.e. 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 provide a way to load and start the query module with separate requests, if this is desired.

Registering a Project via the c8_client Program

You may "register" (compile and run) a project (query module) by giving the c8_client utility the register-ccl-query command. The 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>
```

Parameter	**Description**
config_file	This is the path to the Coral8 Server's configuration file (which is usually named `coral8-server.conf` and is usually found under the `server/conf` directory).
compiler-server-uri	The URL of the manager of the server that you would like to compile this project. Note that 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", e.g. "`http:MyComputer:6789/Compiler`".

manager-server-uri	The URL of the manager server on which you would like to run this project. Note that 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", e.g. "`http:MyComputer:6789/manager`".
path_to_ccl_query_file	The full path and file name of the file that contains the CCL statements. Note that this file should contain *only* CCL statements and nothing else. (Versions of Coral8 Studio up through and including at least version 5.2 create .ccp and .ccl files that include not only CCL statements, but also XML tags. Such files are not suitable for use as the CCL query file with this c8_client command. Note that this may change in future versions.)
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.
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 from running after you have started it.

Compiling and Running a Project

You can compile a project (query module) locally and then run it by giving the c8_client utility the compile-and-run command. The 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>]
```

Parameter	**Description**
config_file	This is the path to the Coral8 Server's configuration file (which is usually named `coral8-server.conf` and is usually found under the `server/conf` directory).
server_uri	The URL of the server on which you would like to run this project. Note that 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", e.g. "`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, with the exception of the workspace and program names.
workspace_name	The name of the workspace in which the query module should be run.
program_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 after you have started it.

Get Status of an Executing Project

After a project has been started on a Server, it is usually necessary to monitor the status of the project. There are two ways to accomplish this:

- 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>
```

(This command should be on a single line. We have split it across multiple lines for readability.)

Publishing Data to a Server

In the overwhelming majority of cases, when you want to publish data to a stream in a server, you will use an adapter. You may, however, use the c8_client program to publish data to a stream. To do this, you 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.

```
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
```

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

In this example,

- The data format is "csv" (Comma-Separated Values).

- The Row Timestamp will be based on the time that the row arrives, not based on a Row Timestamp inside the row.
- The field delimiter character (in other words, 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 will 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 the version of Coral8 Server that you are using, please execute the command:

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

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

Parameter	**Description**
format	Must be one of the three Coral8 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. You need this if user authentication is enabled.
password	The password of the user that you specified. Note that the password should only be specified if a username has been specified.
uri	The URI of the stream to which you want to publish. For instructions on obtaining the URI of a stream, see How to Find the URI of a Stream.
timestamp-mode	The value may be set to "as-is" or "set-current". If the value is "as-is", then the server will use the row timestamp in the data. If the value is "set-current", then the server will replace the row timestamp with the current timestamp (as of the time that the row is received by the server).
csv-title-row	If the value is set to "true", the server will treat the first row as a title row. If the value is set to false, the server will treat the first row as a row of data.
csv-timestamp-column	If this is set to "true", then the first row of data will be treated as the Row Timestamp. If this is set to false, then the data will be treated as though it had no Row Timestamp.
csv-timestamp-column-	If this is set, then the server will assume that the format of all

format	timestamp columns, including the Row Timestamp (if a Row Timestamp is present) matches this format. If this is not set, then the server will assume 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 CCL Reference.
format-options	For a list of format options, see the table that describes Format Options for Publishing with c8_client,

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

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 may be single quotes or double quotes. On MS-Windows, the quotation marks should 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.

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; use "TitleRow" instead.
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; use "TimestampColumn" instead.
TimestampColumnFormat	This is 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 CCL Reference.
NullColumnValue	This is 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	A character that terminates a line of CSV input (in other words, that terminates a row). The default character is a newline character.
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 (") may be 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 backquote character should all be treated as delimiters for STRING fields: `--format-options="QuoteChars`\"'",`
EscapeChars	The escape character is a 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 would not want it to continue to be treated as a whitespace character, so you might create a set of whitespace characters that exclude

	the tab character.
TrimWhitespaces	If this is "true", then whitespace characters outside of quotes will be trimmed from STRING fields. If this is "false", then whitespace characters will be preserved. Whitespace characters inside quotation marks are always preserved.

Stop Execution of a CCL Project

After a CCL project (query module) has been successfully started on the Server, it will continue to execute until it is stopped by a client request. The c8_client command to stop a running project is "stop".

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

(This command should be on a single line. We have split it across multiple lines for readability.)

This stops the CCL application <application-name> in Coral8 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 or --program-name or --load option when the module was compiled.

Clean Up a Workspace's Resources

Once a workspace is no longer being used, it should be destroyed using the manager-destroy-workspace c8_client command. This command takes the same parameters as the manager-create-workspace takes.

```
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. We have split it across multiple lines for readability.)

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 Coral8 Server

This section describes how to stop Coral8 Server.

Stop the Server on UNIX-like Operating Systems

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

```
coral8-server.rc stop
```

The script **coral8-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

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, then you stop the server by pressing ctrl-C in the command window in which the server is running. There is no API or other programmatic mechanism for stopping the server on Microsoft Windows.

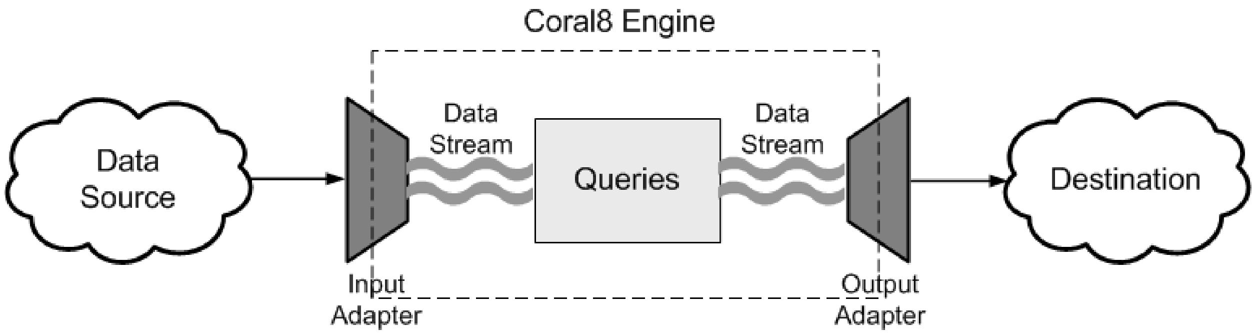
Implementing Guaranteed Processing

Overview

Normally when you run a Coral8 project or application, Coral8 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. In other words, guaranteed processing produces exactly the same results (other than timing) with failures as processing would have produced without failures.

Application Components

When you implement a guaranteed processing application, you must address every component of the application, not just the Coral8 Engine components. The following diagram illustrates the components of an end-to-end application incorporating Coral8 Engine:



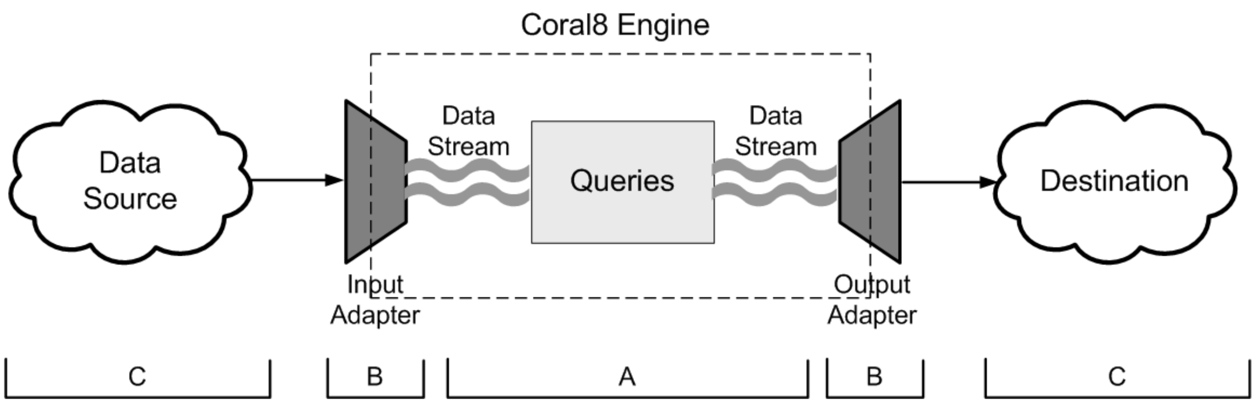
Messages travel through the application as follows:

- 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 performs whatever actions with the data that it is designed to do.

In order for an application to achieve guaranteed processing, every component of the system—every link in the chain—*must* handle the logic and communication with its neighbor components that ensure exactly-once delivery. 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

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



Coral8 Engine

The components labeled with **A** in the diagram—the streams, projects, and query modules—are part of Coral8 Engine and can implement exactly-once processing with the help of the Coral8 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. This usually requires persistence. In addition, to be able to recover from a Coral8 Server failure quickly, you should consider implementing High Availability.

Note that 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 Coral8 Engine Settings 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 Coral8 or one of Coral8's partners that 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 Coral8, see Coral8 Adapters for more information. If you are writing your own adapter, see Writing an Adapter for Guaranteed Processing 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 Coral8 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.

Coral8 Engine Settings

You can use Coral8 Studio or the Coral8 Eclipse plugin to enable Guaranteed Delivery on your projects, modules, and streams. You can also specify Guaranteed Delivery settings with dynamic queries if you are using a Coral8 SDK to build a custom application.

Settings for Projects, Modules, and Streams

You enable and configure Guaranteed Delivery for a project, submodule, or stream in the Properties view of Coral8 Studio. You can also use properties of the CREATE MODULE and CREATE STREAM statements to enable Guaranteed Delivery on a module (in the Coral8 Eclipse plugin only) or stream. For more information about the Properties pane, see the "Editing Query Module Properties" section of the *Coral8 Studio Guide*. For more information about the CREATE MODULE statement, see the documentation provided with the Coral8 Eclipse plugin. For more information about the CREATE STREAM statement, see the section of the same name in the *Coral8 CCL Reference*.

Coral8 query execution is repeatable: providing the same inputs to multiple executions of the same query will produce the same result. This characteristic allows Coral8 Engine to recover from a failure, since it can, in essence, pick up where it left off. Note that there are a few fairly obvious 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

The Guaranteed Delivery protocol by itself doesn't ensure exactly-once processing: Coral8 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. You enable persistence at the project and module level using the same mechanisms described in Settings for Projects, Modules, and Streams. Streams inherit their persistence settings from the parent project or module.

You usually enable persistence on projects and modules whenever you enable Guaranteed Delivery. 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.

For more information about persistence, see the *State Persistence* technical article.

Start with Clean Slate

When you use Coral8 Studio to start a project with persistence enabled, you're essentially restarting the project from the saved state. If you want to start fresh, you instead need to click **Start with Clean Slate** from the Coral8 Studio **Debug** menu. Making any change to the project also clears the saved state, so starting the project after a change is similar to starting with a clean slate.

Writing an Adapter for Guaranteed Processing

If you want to write your own adapter for a guaranteed processing system, you'll need to incorporate the Coral8 Guaranteed Delivery protocol and also handle guaranteed processing communication with your data source or destination.

Guaranteed Delivery Mechanisms

When you write an adapter that incorporates the Coral8 Guaranteed Delivery protocol, you establish a session with Coral8 Engine and send or receive messages in batches. The receiver (subscriber) acknowledges receipt of each batch as the receiver processes it, notifying the sender (publisher) that it is safe to send the next batch of messages.

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

- Establish a connection to Coral8 Engine with a unique session ID, chosen by you.
- Send messages in a batch, each with a batch ID (again chosen by you) unique to the session, and wait for acknowledgment of that batch from the subscriber.
- Detect a return after a failure (a lost connection, for example) and re-establish the connection to Coral8 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 Coral8 Engine with a unique session ID, chosen by you.
- 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 Coral8 Engine with the same session ID. Notify Coral8 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 Coral8 C/C++ SDK, Coral8 Java SDK, and Coral8 .NET SDK.

Note that 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 Coral8 Guaranteed Delivery functions, you must include code in your adapter to handle a variety of situations that may not be 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 Coral8 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 Coral8 C/C++ SDK

Both Coral8 Server and Coral8 Studio ship with the Coral8 C/C++ SDK. All the C/C++ SDK files are in the directory **sdk/c** under the Coral8 Server or Coral8 Studio installation directory.

Note that the discussion in the following pages is limited to functions specific to Guaranteed Delivery. It does not cover general adapter implementation functions. See Adapters and Coral8 C/C++ SDK for more general information about writing adapters.

Publishing for an In-Process Adapter

The following table lists the functionality required to implement Guaranteed Delivery in an input adapter and the corresponding Coral8 C/C++ in-process API functions you use for each:

Establish a session	C8AdapterConnect
Create and publish batches of messages	C8AdapterSendMessageListAsBatch
Re-establish a session after a failure	Coral8 Engine calls the developer-defined reconnect function as specified in the ADL file.
Request the ID of the last batch processed by Coral8 Engine	C8AdapterGetLastBatchId

See In-process Adapter API and Signatures of User Functions for specific syntax and usage information.

Publishing for an Out-of-Process Adapter

The following table lists the functionality required to implement Guaranteed Delivery in an input adapter and the corresponding Coral8 C/C++ out-of-process API functions you use for each:

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 Coral8 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 API Interface for specific syntax and usage information.

Coral8 provides an example input adapter that implements Guaranteed Delivery, called **example_gd_input_adapter.c**. You can find it under your Coral8 Server or Coral8 Studio installation directory, in the **sdk/c/examples** directory.

Subscribing for an In-Process Adapter

The following table lists the functionality required to implement Guaranteed Delivery in an output adapter and the corresponding Coral8 C/C++ in-process API functions you use for each:

Establish a session	C8AdapterConnect
Receive and process batches of messages	C8AdapterGetNextMessagesBatch C8MessageBatchPopMessage
Re-establish a session after a failure	Coral8 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, Signatures of User Functions, and Message API for specific syntax and usage information.

Subscribing for an Out-of-Process Adapter

The following table lists the functionality required to implement Guaranteed Delivery in an output adapter and the corresponding Coral8 C/C++ in-process API functions you use for each:

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

Note that 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 API Interface for specific syntax and usage information.

Coral8 provides an example output adapter that implements Guaranteed Delivery, called **example_gd_output_adapter.c**. You can find it under your Coral8 Server or Coral8 Studio installation directory, in the **sdk/c/examples** directory.

Guaranteed Delivery with the Coral8 .NET SDK

Both Coral8 Server and Coral8 Studio ship with the Coral8 .NET SDK. All the .NET SDK files are in the directory **sdk/net3** under the Coral8 Server or Coral8 Studio installation directory. The SDK reference documentation is in the directory **sdk/net5/doc**. Open the file **Documentation.chm** to get started.

Coral8 ships several examples with the .NET SDK, which are under **sdk/net3/examples**. The examples are thoroughly commented to help you understand the purpose of each line of code. For more information about examining, compiling, and running the Coral8 .NET examples, see Coral8 .NET SDK.

The file named **Example_13_WorkingWithGuaranteedDelivery.cs** contains an example illustrating how to use the Guaranteed Delivery protocol with the Coral8 .NET SDK.

Note that the discussion in the following sections is limited to methods specific to Guaranteed Delivery. It does not cover general adapter implementation methods. See Adapters and Coral8 .NET SDK for more general information about writing adapters.

Publishing

The following table lists the functionality required to implement Guaranteed Delivery in an input adapter and the corresponding Coral8 NET SDK methods you use for each:

Establish a session	CreatePublisherWithGuaranteedDelivery
Create and publish batches of messages	NewBatchOfMessages publish
Re-establish a session after a failure	CreatePublisherWithGuaranteedDelivery
Request the ID of the last batch	GetLastBatchId

processed by Coral8 Engine	

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 lines of the example creates a publisher object with Guaranteed Delivery enabled and establishes 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 to this method 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 Coral8 Engine processed:

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

Now the publisher knows where to restart processing (which batch of messages it needs to send).

Subscribing

The following table lists the functionality required to implement Guaranteed Delivery in an output adapter and the corresponding Coral8 .NET SDK methods you use for each:

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 Coral8 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 Coral8 Engine which batch to send next.

Variations of Guaranteed Processing

A guaranteed processing application incorporates both Guaranteed Delivery and persistence. Depending on your application needs, you might want to enable just one of those features in Coral8 Engine.

Guaranteed Delivery without Persistence: If you enable Guaranteed Delivery for the Coral8 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 Delivery: If you enable persistence for the Coral8 Engine components of your application without enabling Guaranteed Delivery, the application will still perform best-effort processing, but if Coral8 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.

Coral8 C/C++ SDK

This chapter explains how to do the following in C/C++:

- Create an out-of-process adapter
- Create an in-process adapter
- Register a query
- Compile a CCL project
- Create a User-Defined Function
- Control the server - e.g. start or stop a query module, or get status information about a server or a query module.
- Embed a Coral8 Server inside another process

Overview

The Coral8 SDK (Software Development Kit) in the 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 Coral8 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.

The only C compiler that Coral8 supports on Microsoft Windows is the Microsoft Visual Studio .NET 2005 C/C++ compiler with Visual Studio Service Pack 1.

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 Coral8 Server. You should re-compile the code. This applies to all .dll files that are intended to be linked in with Coral8 Server, including:

- in-process adapters
- UDFs (User-Defined Functions). 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 (Remote Procedure Call) plugins
- user-defined plugins

Compiling for 64-bit Microsoft Windows

When compiling for 64-bit Microsoft Windows, do the following:

- 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".

     If you don't see "x64" in the list, then choose "<New...>"

     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

In chapter 3, we explained the difference between an in-process adapter and an out-of-process adapter. The concepts of "in-process" and "out-of-process" apply to more than just adapters.

Compiling, registering a query, and server control (e.g. stopping execution of a project) are all out-of-process activities. All of these out-of-process activities use the Coral8 client SDK.

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 Coral8 Server SDK.

Data Types and Subroutines for UDFs and In-process Adapters

Coral8 provides some functions that can be called from either User-Defined Functions or In-process Adapters. These functions are described in this section. We suggest that you skim this section the first time you read it. Later, after you have learned more about how to do a particular task (create a UDF or adapter), return to this section and read it in more detail.

Datatype is one of the internal forms of data used by the Coral8 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 bits).

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 (Binary Large OBject).

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 ".

Note that BLOB data may be stored in one of 3 formats:

- raw - a "raw" BLOB is stored in its original format, i.e. as a byte-for-byte copy of its value. This is the most compact form, and is the form in which Coral8 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 must not 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.

Each Coral8 C/C++ SDK function that uses hex string format (either for input parameters or return values) will contain "HexString" somewhere in the name, e.g. 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

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 single call. Ignoring error conditions may result in undefined behavior.

Users may implement their own error code with associated text messages. This is an alternative way to log errors on the Coral8 engine logger. All calls to C8ErrorSet() result in a log entry in the Coral8 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 Coral8 API function is called. (Each Coral8 API call will reset the error.)

A zero error code indicates no errors; this is the initial state.

Refer to c8messages.h for Coral8 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 Coral8 Server log.

The err_text variable may contain not only text, but also any of the print format specifiers allowed by the NSPR C-language printf() function (e.g. "%s", "%d", etc.). 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 newline to force each message to be put on a separate line in the Coral8 log.

The table below shows the print format specifiers used with Coral8 data types.

Coral8 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 up error numbers and error messages so that the user sees the same error message each time she sees a particular error number (and vice versa). However, Coral8 does not require or check that error codes and error messages are consistently paired up. In different calls, the same error code may be used with different text and vice versa.

Note that the error that you set with C8ErrorSet() will normally remain until you explicitly clear it with the `C8ErrorClear()` function or until it is written over by another error.

Note that errors are "per thread". Each thread sees only its own error codes.

Calls to the Coral8 API will clear any error code when they succeed and set the error code when they fail. We recommend that users always check the return codes of Coral8 calls to make sure they succeeded.

- 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 Coral8 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 Coral8 Server log.

The log_text variable may contain not only text, but also any of the print format specifiers allowed by the NSPR C-language `printf()` function (e.g. "%s", "%d", etc.). 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 newline to force each message to be put on a separate line in the Coral8 log.

Parameters:

- log_text - The text to write to Coral8 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 some 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 coral8-server.conf file, controls whether the message is sent to the log. For example, if coral8-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 may contain both text and any of the print format specifiers allowed by the NSPR C-language `printf()` function (e.g. "%s", "%d", etc.). 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 newline to force each message to be put on a separate line in the Coral8 log.

Parameters:

- i_code - This is a number that identifies the type of error that occurred.
- i_level - 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: Return 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 current error code (for the current thread). This length takes into account 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. (Coral8 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 current 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 the user has properly 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

The following memory management functions may be used with either in-process operations or 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 (i.e. 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.

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.

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.

Parameters:

- mem_ptr - a pointer to the memory to be free'd.

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 section below.

Notes about 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, then 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 you should free the memory 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 will be responsible for de-allocating, you must allocate the memory using the appropriate function so that the server will de-allocate it correctly.

These rules apply to all code in In-process Adapters and User-Defined Functions (including user-defined aggregate functions).

The following table shows when to use each type of de-allocation method:

Allocated By	**Deallocated By**	**Language**
`C8Malloc() / C8Realloc()`	`C8Free()`	C or C++
`malloc() / 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++

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 an 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 Coral8 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 - e.g. 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 "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, you should include the appropriate library as part of the link command.

Datatype ToString()

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 Coral8 default format, pass NULL as the format argument. The user is 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 Coral8 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). Users 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.
- 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 - 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.
- 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 into which to store the string. The user must have allocated enough memory to hold the desired string.

- 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, conversion may fail.

Parameters:

- i_value - the value to convert.
- i_format - a format specifier indicating the desired format of the resulting string value, e.g. "YYYY-MM-DD HH24:MI:SS.FF". If the format parameter is a null pointer, then the output is in microseconds. To specify another format, please refer to the CCL Reference Guide's description of valid formats for TIMESTAMP data type. If the time zone is not specified, then the format will default to using local time.
- o_buf - the location into which to store the string. The user must have allocated enough memory to hold the desired string.
- 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 Coral8 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, e.g. "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, e.g. "PST" for Pacific Standard Time.
- o_buf - the location into which to store the string. The user must have allocated enough memory to hold the desired string.
- o_buf_size - 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 Coral8 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 into which to store the string. The user must have allocated enough memory to hold the desired string.
- 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 into which to store the string. The user must have allocated enough memory to hold the desired string.
- o_buf_size - 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 2 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 into which to store the string. The user must have allocated enough memory to hold the desired string.
- 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 may 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 Coral8, 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.

```
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, then the output is in microseconds. To specify another format, e.g. "YYYY-MM-DD HH24:MI:SS.FF", please refer to the CCL Reference Guide's description of valid formats for TIMESTAMP data type. If the format is invalid, conversion may fail.

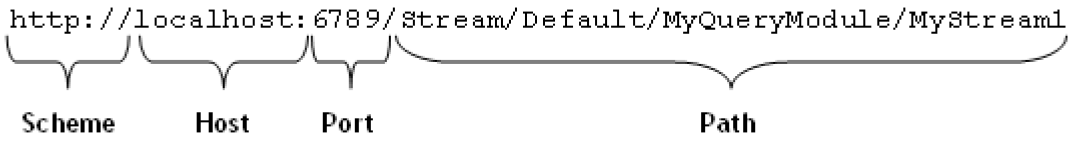
Note that since 2 hex digits are converted to a single byte in the BLOB, and since the trailing null (string terminator) is not needed as part of the blob, the o_out parameter to the `C8ConvertHexStringToBlob()` function only needs to be CEIL((N-1)/2) bytes long, where N is the length of the hex string.

Note also that C8SizeType is typedef'd in c8types.h.

Miscellaneous

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 URL of the form:


http://localhost:6789/Stream/Default/MyQueryModule/MyStream1
Scheme
Host
Port
Path

parse the URL into the various components:

- scheme - the protocol, e.g. "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. The user 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 (Uniform Resource Locator) that you would like parsed.
- scheme - the scheme specifies the protocol, e.g. "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 path is 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

C8Timestamp C8Now();

Purpose: This returns the current time. The returned value is a standard TIMESTAMP value; i.e. 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 resolution 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; i.e. 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.

Note that this function is considerably more "expensive" than C8Now().

Parameters: none.

Returns: the current time as a C8Timestamp.

void C8Sleep(C8Interval us_delay);

Purpose: The thread sleeps for the specified amount. The header file c8types.h contains definitions that may be useful in specifying durations; e.g., to delay 3 seconds, use 3*C8PerSecond.

Note that small 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 almost certainly 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 API(s) are used both by out-of-process adapters (i.e. 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

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.

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 containing security certificate information. Simply copy the **secure** directory into the same directory as your executable from the top-level directory of either the Coral8 Server or Coral8 Studio installation, first making sure that the Coral8 application is not running. If you fail to take 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 if successful, C8_FAIL otherwise.

APIs Used for In-process and Out-of-process Adapters

The following APIs may be used with both in-process and out-of-process adapters.

API Interface

Coral8 provides some header (.h) files that are used in both in-process and out-of-process adapters.

The header file c8types.h provides the data type information.

The shared APIs include:

- **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. The function prototypes for the schema API are in the file c8messages.h
- **Message**. Data routing in the Coral8 engine is in terms of data packets called "messages" (informally also called "rows" or "tuples"). A message contains, among other things, a particular 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. For Guaranteed Delivery, messages are sent in batches. The function prototypes for the message and message batch APIs are in the file c8messages.h

Schema API

This section describes the schema-related APIs in the header file c8messages.h.

C8UInt C8SchemaGetColumns(const C8Schema *i_s);

Purpose: For a given schema, return 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, return the type of information contained in that column. The type information may be used in a switch statement or other control code. Column indexes are 0-based, not 1-based -- i.e. 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 may be used in a switch statement or other control code. Column indexes are 0-based, not 1-based -- i.e. 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. This may be useful in printing titles, etc.

Parameters:

- ptr - a pointer to a schema object.
- col_num - the index of the column for which you want the information.

Returns: the name of the column. The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not 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. Column names are 0-based, not 1-based.

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 may 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 to which you wish to add a column.
- i_type - the data type the new column should have.
- i_name - the name that the new column should have.

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 be a file that contains a compiled schema (i.e. 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.

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 returned pointer must be freed with a call to C8SchemaDestroy(). The input parameter is the string to read the schema from. The stream must contain the XML text representation of a compiled schema (i.e. 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 (de-allocates 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, i.e. a tuple descriptor). This function allocates enough memory for the string, and the pointer returned through the o_buf parameter needs to be free'd 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 2 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

This section describes the message-related APIs in the header file c8messages.h.

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. I.e. it corresponds to a row of data in the stream.

The other types of messages are not currently intended for use with adapters.

Note that, in general, when a "const *" is returned for data associated with an object (e.g. a schema associated with a message), the validity of the returned pointer is limited to the lifetime of the pointer's "owner" (e.g. the message). Attempts to de-reference the pointer after the "owner" no longer exists may, of course, 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 will be 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, you must destroy it by passing it to `C8MessageDestroy()`.

See also the function C8MessageCreateWithSize().

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 specified amount of memory allocated. This is the initial call in composing a message. By allowing the user to specify the size of the entire message, performance may be increased.

Parameters:

- msg_type - in almost all cases, this will be 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.

- field_size - A user estimated size of the entire message.

Returns: a pointer to the newly created message. When you are done using this message, you must destroy it by passing it to `C8MessageDestroy()`.

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 a value that requires 5 bytes, then the total size for that field will be 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.

See also the function C8MessageCreate().

void C8MessageDestroy(C8Message *msg_ptr)

Purpose: destroys the message pointed to by msg_ptr. If a message has been created, it must be destroyed after it has been dispatched to the Coral8 Server. All messages created by the user or received by the C8AdapterReceiveMessage() or C8AdapterReceiveMessageWait() call must be destroyed or 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 may arrive in "bundles" (see the definition of bundle earlier in this manual). 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, then you can recognize the start and end of a bundle by recognizing these messages. If you don't need to deal with bundled messages differently from the way that 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 *Coral8 CCL Reference* 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 and the user wishes 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. The user should not change the memory that the pointer points to, and the user should 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:

- 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. Users should 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. Users should 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 that 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. 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.

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 an index of a column in that message, return the value of the column in the form of a string. o_res will be 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 will be 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 for which to get the data.
- 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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. Note: The user must **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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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. Note: The user must **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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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 returned 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.

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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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.

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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`.

Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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. Note: The user must **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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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. Note: The user must **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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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.

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, a caller should ensure that the desired data type is indeed stored at col_ndx by calling `C8SchemaGetColumnType()`. Before retrieving each row's value, a caller should 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the pointer named "value" was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after calling `C8AdapterSendMessage()` or a memory leak will result. (The `C8MessageColumnSetString()` routine will make a copy of the value, so freeing that value will 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the i_buf pointer was obtained through `C8Malloc()`, the user must free the pointer (via C8Free()) after C8AdapterSendMessage() or a memory leak will result. (The `C8MessageColumnSetBlob()` routine will make a copy of the value, so freeing that value will 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.)
- 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.
- 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the i_base64_string pointer was obtained through `C8Malloc()`, the user must free the pointer (via C8Free()) after C8AdapterSendMessage() or a memory leak will result. (The `C8MessageColumnSetBlob()` routine will make a copy of the value, so freeing that value will 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.

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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after `C8AdapterSendMessage()` or a memory leak will result. (The `C8MessageColumnSetXmlFromString()` routine will make a copy of the value, so freeing that value will 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after `C8AdapterSendMessage()` or a memory leak will result. (The

`C8MessageColumnSetStringByName()` routine will make a copy of the value, so freeing that value will 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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via `C8Free()`) after `C8AdapterSendMessage()` or a memory leak will result. (The `C8MessageColumnSetBlobByName()` routine will make a copy of the value, so freeing that value will 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.

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 will be logged. If the col_ndx is out of range, the set will be ignored and an error message logged.

If the value pointer was obtained through `C8Malloc()`, the user must free the pointer (via C8Free()) after `C8AdapterSendMessage()` or a memory leak will result. (The `C8MessageSetXmlFromStringByName()` routine will make a copy of the value, so freeing that value will 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.

The user is responsible for C8Free()'ing the returned string!

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 Coral8 message. Currently the only format supported is C8_MESSAGE_XML. The user is responsible for C8Destroy()'ing the returned message.

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 about which you want the count.

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 through 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++

This chapter explains how to write your own out-of-process adapter in the C/C++ programming language. You may write either an input adapter or an output adapter this way.

Coral8 provides a set of functions that allow you to attach to (connect to) a stream, read or write data, and detach from the stream. These functions are referred to as the Coral8 C Adapter library.

Coral8 provides:

- A library for resolving a CCL stream URI.
- A library for transferring data via a network.
- Documentation.

Coral8 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, if you installed the Coral8 Engine to the default location, the directory is:

```
C:\Program Files\Coral8\Server\sdk\c\examples
```

On UNIX-like operating systems, if you installed the Coral8 Engine to "/home/<userid>", the directory is

```
/home/<userid>/coral8/server/sdk/c/examples
```

API Interface

As we mentioned earlier, a typical out-of-process adapter performs the following tasks:

1. Acquire an "address" (URI) that will uniquely identify a specific stream and tell the communication layer (provided by Coral8) how to find that stream.
2. Open a connection to the stream.
3. Read (or write) the desired data. Typically the read (or write) operation is in a loop.
4. Close 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 sending (publishing) messages.

Parameters:

- i_uri - - the URI of the stream to which you wish to publish.

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 to which you wish to publish.
- i_credentials - - a pointer to a structure containing the user's 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 for information about Guaranteed Delivery) and establishes (or re-establishes) a session with Coral8 Server. The returned publisher object is used later when sending (publishing) messages.

Parameters:

- i_uri -- the URI of the stream to which you wish to publish.
- i_credentials -- a pointer to a structure containing the user's credentials (user name 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_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. The user 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 as a batch all messages that have been queued with **C8PublisherSendMessage** or **C8PublisherSendMessages**. 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 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 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 by the server 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: create a subscriber to a given stream.

Parameters:

- i_uri is the URI of the stream to subscribe to.

Returns: Returns a pointer to a subscriber object. Returns NULL if error. Subscribers should be freed with C8SubscriberDestroy().

C8Subscriber *C8SubscriberCreateA(const C8Char *i_uri, const C8UserCredentials *i_credentials)

Purpose: create a subscriber to a given stream that requires user authentication.

Parameters:

- i_uri is the URI of the stream to subscribe to.
- i_credentials - - a pointer to a structure containing the user's credentials (username and password).

Returns: Returns a pointer to a subscriber object. Returns NULL if error. 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: create a Guaranteed Delivery subscriber to a given stream (see Implementing Guaranteed Processing for information about Guaranteed Delivery) 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 the user's 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.

Returns: Returns a pointer to a subscriber object. Returns NULL if error. Subscribers should be freed with C8SubscriberDestroy().

C8Message * C8SubscriberGetNextMessage(C8Subscriber *i_sub, C8Interval i_to)

Purpose: get 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()`.

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 subscriber. For use with a Guaranteed Delivery subscriber.

Parameters:

- i_sub - the Guaranteed Delivery subscriber object from which to get the next batch of messages.
- o_msgs - 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 Coral8 adapter library code (which is linked into the user's out-of-process adapter) and therefore will be running in a thread that is running the Coral8 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 the user's code to check the flag, retrieve the message, and then process the message when their thread executes.

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 is the new callback function for this subscriber.
- i_db_prm is 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 destroy (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. Returns NULL if error. The returned schema remains valid while the subscriber remains valid.

C8Char * C8ResolveUri(const C8Char * i_uri);

Purpose: This resolves a CCL URI into an HTTP/HTTPS URI.

Parameters:

- i_uri - the URI to resolve.

Returns: the HTTP URL. This returns NULL if there is an error.

The pointer returned must be freed with C8Free().

C8Char * C8ResolveUriA(const C8Char * i_uri, const C8UserCredentials *i_credentials);

Purpose: This resolves a CCL URI into an HTTP/HTTPS URI, requiring that the user pass appropriate user authentication credentials before doing so.

Parameters:

- i_uri - the URI to resolve.
- i_credentials - a pointer to a structure containing the user's credentials (user name and password).

Returns: the HTTP URL. This returns NULL if there is an error.

The pointer returned must be freed with C8Free().

C8Char ** C8ResolveClusterUri(const C8Char * i_uri);

Purpose: This 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 there is an error.

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: This resolves a CCL URI into one or more HTTP/HTTPS URIs, requiring that the user pass appropriate user authentication credentials before doing so.

Parameters:

- i_uri - the URI to resolve.
- i_credentials - a pointer to a structure containing the user's credentials (user name and password).

Returns: a pointer to a NULL-terminated array of string pointers, or NULL if there is an error.

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 there is an error.

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 there is an error.

The pointer returned must be freed with C8Free().

Creating a Sample Input Adapter

This section provides a detailed method to connect a simple adapter for writing to Coral8 Server.

To create and run our demo, we use the following files:

- example_input_adapter.c - This file is a very basic input adapter. Like any input adapter, it 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
  ```

E.g. on Microsoft Windows, the directory is typically:

```
C:\Program Files\Coral8\Server\sdk\c\examples
```

On UNIX-like operating systems, the directory is typically

```
/home/<username>/coral8/server/sdk/c/examples
```

Sample Input Adapter

The example shows how to write data to Coral8 Server.

This example generates its own data; when you write your own adapter, you will probably read the data from a data source.

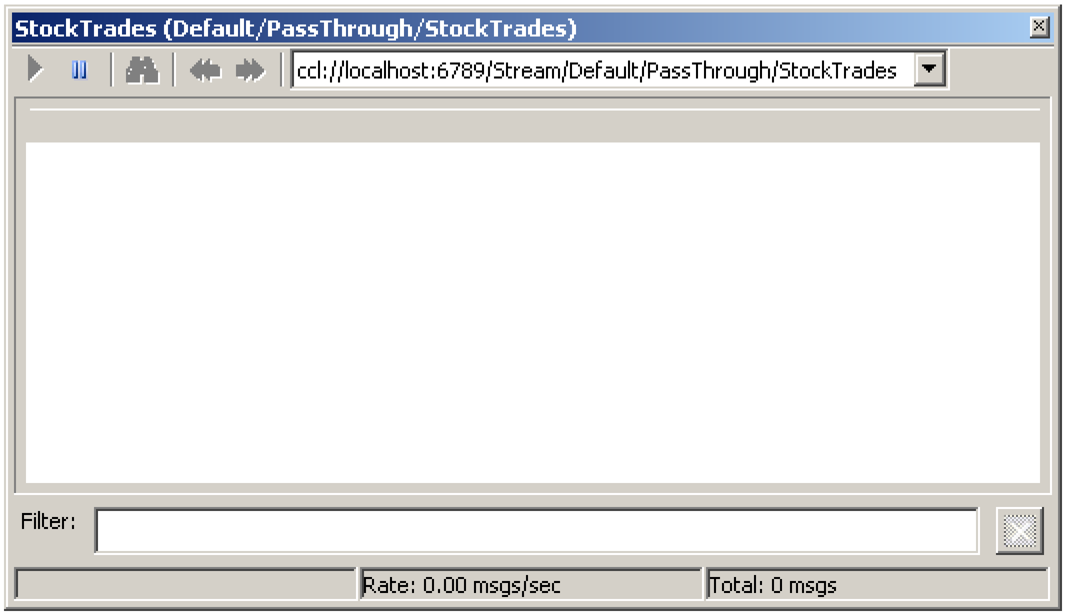
After the normal declarations and command line argument parsing, the input stream is opened. Notice the return status is examined for success.

Acquiring the Address (URI) of a Stream

One open issue remains: acquiring the "address" of the stream so that you can pass it to the function that opens a connection to the stream. The address is a URI. To get the URI of a particular stream, you must first create the stream in Coral8 Studio, then open a stream viewer (right-click on the stream and click on the "view" command), then read the URI shown in the title of the stream viewer window. The URI will look somewhat 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:


StockTrades (Default/PassThrough/StockTrades)
ccl://localhost:6789/Stream/Default/PassThrough/StockTrades
Filter:
Rate: 0.00 msgs/sec
Total: 0 msgs

The URI of the stream will not normally 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 URIs*.

If your out-of-process adapter will send data to multiple streams (for use with the Parallel Query feature), see the section titled "Parallel Queries and URIs" in that same appendix.

Compiling and Linking the Example

The following instructions show you how to compile and link the example code on Microsoft Windows and on UNIX-like operating systems.

Compiling and Linking on Microsoft Windows

This section explains two ways to compile and link on Microsoft Windows:

- from inside Visual Studio, or
- from the command line.

Both sets of instructions assume that you are using Microsoft Visual Studio 2005.

Compiling Using Visual Studio

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 in which you'd like the project to be stored.

   E. Click OK.

   F. The next window to appear will be the "Win32 Application Wizard" window.

   G. Click on "Finish".

3. Microsoft Visual Studio will create a simple C++ (.cpp) file to use as a starting point. We recommend that you remove all the contents of this file and then insert your own C/C++ code for the adapter.

   Make sure that 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. 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 "MySample 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 directories listed below.

      On 32-bit Microsoft Windows, the directories are typically:

```
C:\Program Files\Coral8\Server\sdk\c\include\
C:\Program Files\Coral8\Server\sdk\c\include\nspr
```

      On 64-bit Microsoft Windows, the directories are typically:

```
C:\Program Files (x86)\Coral8\Server\sdk\c\include
C:\Program Files (x86)\Coral8\Server\sdk\c\include\nspr
```

      You may also add other directories if necessary for your adapter.

To add these, do the following:

a. Click in the "Additional Include Directories" field, and paste in the first path.

b. Then 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.

Repeate step "b" as many times as necessary to add any remaining required paths.

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 Coral8 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 Coral8 library.

On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\Coral8\Server\sdk\c\lib
```

On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\Coral8\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 C/C++ 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 C/C++ 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\Coral8\Server\sdk\c\include\nspr"
    /I "C:\Program Files\Coral8\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 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\TESTA
dapter1.2.exe"
  /INCREMENTAL /NOLOGO /LIBPATH:"C:\Program
Files\Coral8\Server\sdk\c\lib"
  /MANIFEST
/MANIFESTFILE:"Debug\TESTAdapter1.2.exe.intermediate.manifest"
  /DEBUG
  /PDB:"c:\c8test\e3\sdkc\testadapter1.2\testadapter1.2\debug\TES
TAdapter1.2.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 should be 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\Coral8\Server\bin
C:\Program Files\Coral8\Server\sdk\c\lib
```

Compiling Using the Command Line

The following instructions show 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. Make sure that you have configured your C compiler so that you can compile and link at least a simple "hello world" C program. For a sample script that sets environment variables, see the sample script later in this section.

   To ensure that you can access the MS Visual Studio 2005 resources (e.g. library files), you will typically need to set the environment variables PATH, INCLUDE, and LIB appropriately, which you can do 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 Coral8 directories:

   ```
   C:\Program Files\Coral8\Server\sdk\c\include\
   C:\Program Files\Coral8\Server\sdk\c\include\nspr
   ```

3. Update your LIB environment variable to include the following Coral8 directory(s):

   ```
   C:\Program Files\Coral8\Server\sdk\c\lib
   ```

   so the compiler/linker can find the required .lib files when linking.

   To do this, execute commands similar to the following:

   ```
   set LIB=%LIB%;C:\Program Files\Coral8\Server\sdk\c\lib
   ```

4. When you execute the program, make sure that your path includes the following files

   ```
   C:\Program Files\Coral8\Server\sdk\c\lib
   C:\Program Files\coral8\server\bin
   ```

   so the program can find the .dll (Dynamic Link Library) files or .so (Shared Object) files at runtime.

   To do this, execute commands similar to the following:

   ```
   set PATH=%PATH%;C:\Program Files\Coral8\Server\sdk\c\lib
   set PATH=%PATH%;C:\Program Files\Coral8\Server\bin
   ```

5. On Microsoft Windows, execute "cl" (Compile and Link) commands similar to 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 Coral8 .h files
rem --- and the NSPR .h files.
set INCLUDE=C:\Program Files\Coral8\Server\sdk\c\include;%INCLUDE%
set INCLUDE=C:\Program Files\Coral8\Server\sdk\c\include\nspr;%INCLUDE%
rem --- Set the env vars to enable us to find the Coral8 library files
set LIB=C:\Program Files\Coral8\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\Coral8\Server\sdk\c\include\nss\public;%INCLUDE%
rem --- Make sure that the PATH contains the .DLL files.
set PATH=%PATH%;c:\Program Files\Coral8\Server\bin
set PATH=%PATH%;c:\Program Files\Coral8\Server\sdk\c\lib
```

Compiling and Linking on UNIX-like Operating Systems

The following instructions show how to compile and link the example code on UNIX-like operating systems.

1. Make sure that you have configured your C compiler (e.g. gcc) so that you can compile and link at least a simple "hello world" C program.

   You will typically need to set the environment variables PATH, INCLUDE, and LIB appropriately.

2. Update your LIB environment variable to include the proper library directory. If you installed the Coral8 Engine under the directory "/home/<userid>" then the paths will be

   ```
   /home/<userid>/coral8/server/sdk/c/lib
   ```

   so the compiler/linker can find the required .lib files when linking.

   To do this, execute commands similar to the following:

   export LIB=${LIB}:/home/<userid>/coral8/server/sdk/c/lib

3. Make sure that your PATH environment variable includes the following paths

   ```
   /home/<userid>/coral8/server/sdk/c/lib
   /home/<userid>/coral8/server/bin
   ```

   so the program can find the .so (Shared Object) files at runtime.

   To do this, execute commands similar to the following:

   ```
   export PATH=${PATH}:/home/<userid>/coral8/server/sdk/c/lib
   export PATH=${PATH}:/home/<userid>/coral8/server/bin
   ```

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. You will need to customize this file to specify the userid. If you did not install the Coral8 Engine in "/home/<userid>" then you will need to make additional customizations, also. The example below 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 "(e.g. with the '.' command) "
echo "rather than simply run it."
# Include the standard Coral8 C/C++ SDK files.
# We'll use this environment variable later when we compile.
export INCLUDE1=/home/<userid>/coral8/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>/coral8/server/sdk/c/include/nspr
export LIB1=/home/<userid>/coral8/server/sdk/c/lib
export LIB2=/home/<userid>/coral8/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. You may need to 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

These instructions assume that you have already installed the Coral8 Server and Coral8 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 Studio.
3. Start Coral8 Studio, connect to the Default workspace, and load the PassThrough example, located in the Coral8 repository under **examples/FeatureExamples/Basic/PassThrough**.
4. Click on the '+' icon next to the PassThrough to show the Streams.
5. Click on the '+' icon next to the InTrades stream to view the existing adapter. (optional)
6. Open stream viewers for both InTrades and OutTrades. (You can open a stream viewer 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 should start displaying values.
8. Start the input adapter. Note that you should start the input adapter LAST. 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++

In-process adapters are compiled into dynamically linkable libraries (called "shared object" libraries on most UNIX-like operating systems). These libraries are linked into the server, and the code in them runs as part of the server process.

This API allows you to write both input adapters and output adapters.

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.

This chapter shows how to create simple adapters for reading from and writing to a Coral8 Server in the C and C++ programming languages. Since this example is provided in source form, modifications are easily performed.

For an adapter to function, it is sufficient that it knows how to attach to its Coral8 stream and convert data to (or from) the Coral8 Server representation. These two tasks -- communications 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 Coral8 Server. The user must write conversion functions that are appropriate for the source (or destination) of the data.

The Components of an In-process Adapter

This in-process adapter must be linked with Coral8 Server and with SDK libraries. This 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 (Adapter Definition Language) 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. The user must write an ADL file for each in-process adapter and place it in both the Studio plugins directory and the Server plugins directory. This is necessary since the Coral8 Studio and Server may execute on different machines across a network.

Users must also place a library containing their in-process adapter code into the bin folder of the server. When the Coral8 Server determines a user adapter must be loaded, the Coral8 Server code will load the user written adapter ADL, find the name of the library and entry points in the library, load the library and begin executing user-written code.

Algorithm Overview

The C/C++ subroutines in an in-process adapter run only when called by Coral8 Server. Each in-process adapter must provide three or four subroutines that can be called by the server:

- initialize()
- execute()
- shutdown()
- reconnect() (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 the user; for convenience, we will refer to them as the initialize(), execute(), and shutdown() functions.

The initialize() function is called once. In this function, the user does whatever initialization steps are required. For example, if the adapter is an input adapter, then the initialize() routine will typically open or connect to the data source. If the data source is a file, then the initialize() routine will open the input file. The Initialize routine may also read parameters (such as the location of the input, e.g. the directory path in which the input file resides), allocate memory, etc.

The `execute()` function is called periodically. If the adapter is an input adapter, the `execute()` function will read one or more rows from the data source, do any required conversion of the data, and then send that data to the stream. If the adapter is an output adapter, the `execute()` function will read one or more rows from the data stream, do any required conversion of the data, and then send that data to the data destination. Note that when the `execute()` function is called, there is not necessarily any data to process. 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 memory, etc.

When each 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 may have multiple instances of a user-created adapter running simultaneously, just as you may have multiple copies of a built-in adapter running simultaneously.) This adapter pointer is required for many of the calls to the Coral8 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.

Naturally, each of these functions may call other functions written by the user and various library routines that are part of the server or the In-process Adapter SDK. The library file that contains the user's in-process adapter will typically have not only the 3 main routines, but also all the other routines that the user has written in support of those main routines.

Session State Information and Persistent State Information

An adapter's execute() function is typically 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, then after you open that output file you will want to store a file pointer to that file so that 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 Coral8 C/C++ SDK provides functions that allow you to store and retrieve session state. Note that 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

Coral8 provides several header (.h) files. These files help users to declare variables as Coral8 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 earlier.
- **Message**. Data routing in the Coral8 engine is in terms of data packets called "messages" (informally also called "rows" or "tuples"). A message contains, among other things, a particular 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 earlier.
- **In-process Adapter**. Each adapter provides an access point for a message to enter or exit the Coral8 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 or all in-process adapter code should include:

```
#include "c8types.h"
#include "c8adapter_in_process.h"
#include "c8conversions.h"
#include "c8server.h"
```

Coral8 also provides a set of functions that convert between strings and Coral8's internal data types (such as TIMESTAMP, FLOAT, etc.)

Memory Management API

The memory management API is the same for in-process and out-of-process operations. See Memory Management API. Some special cautions apply when these are used with in-process operations. See Notes about Allocating and Deallocating Memory in In-process Code.

In-process Adapter API

These in-process adapter API signatures are defined in c8adapter_in_process.h.

Each adapter gets executed in its own thread. All adapter callbacks (initialize(), execute(), and shutdown()) are **always** called from this thread.

For input adapters, the Coral8 engine will call the execute() callback function as often as possible.

For output adapters, the Coral8 engine will call the execute() callback on a periodic basis. The user should check for message availability. There may 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 Coral8 engine.

const C8Schema *C8AdapterGetSchema(C8Adapter *i_aptr)

Purpose: given an adapter pointer, return the associated schema. The schema specifies the name and the data type for each of the columns. The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not 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* for detailed descriptions of how to use ADL files. All the GetParam names are displayed on the adapter Properties View in Coral8 Studio and are user configurable. All parameters may access the original string form as displayed in Coral8 Studio by calling C8AdapterGetParamString(). If a user has additional data types C8AdapterGetParamString() is the appropriate function call. After obtaining the string, the user may call one of the data conversion functions supplied by Coral8.

The "name" string should be the identical string as 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". Case is significant. Using an unknown "name" results in the "default" value being returned.

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. 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, then returns the value passed as **default**.

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, the default value as specified in the ADL file, or the default value passed as a parameter.

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. 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, then returns the value passed as **default**.

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, the default value as specified in the ADL file, or the default value passed as a parameter.

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. 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, then returns the value passed as **default**.

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, the default value as specified in the ADL file, or the default value passed as a parameter.

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. 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, then returns the value passed as **default**.

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, the default value as specified in the ADL file, or the default value passed as a parameter.

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. 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, then returns the value passed as **default**.

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, the default value as specified in the ADL file, or the default value passed as a parameter.

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. 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, then returns the value passed as **default**.

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, the default value as specified in the ADL file, or the default value passed as a parameter.

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. 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, then returns the value passed as **default**. The parameter "name" may 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() or a memory leak *will* occur.

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, or the default as specified in the ADL file, or the default value passed as a parameter. The returned pointer *must* be released by C8Free() or a memory leak *will* occur.

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. 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, then returns the value passed as **default**. This can be convenient for user-defined data types. The returned pointer *must* be released by C8Free() or a memory leak *will* occur.

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, or the default value specified in the ADL file, or the default value passed as a parameter. The returned pointer *must* be released by C8Free() or a memory leak *will* occur.

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.

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. 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, then returns the value passed as **default**. This can be convenient for user-defined data types. The returned pointer *must* be released by C8Free() or a memory leak *will* occur.

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, or the default value specified in the ADL file, or the default value passed as a parameter. The returned pointer *must* be released by C8Free() or a memory leak *will* occur.

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.

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 Coral8 Server 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 this adapter. If no messages are available, a null pointer is returned.

When the user has finished processing the message, the user should 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()` callback function should return to Coral8 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 this 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 this adapter. If no messages are available, the call will wait for a maximum of "interval" microseconds. If no message is available within "interval" microseconds, then a null pointer is returned. Callers should check the return value to see whether a message arrived or whether a null pointer was returned.

When the user has finished processing the message, the user should call `C8MessageDestroy()` to release message resources. The user-written `execute()` function should continue to process messages as long as `C8AdapterReceiveMessageWait()` (or C8AdapterReceiveMessage()) provides them. When a null is returned from `C8AdapterReceiveMessageWait()`, the `execute()` callback function should return to Coral8 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 this adapter. Returns NULL if there are no new messages.

C8MessageBatch* C8AdapterGetNextMessagesBatch(C8Adapter* adapter, C8Interval i_timeout);

Purpose: retrieves the next batch of messages from Coral8 Server. 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.

Returns: a pointer to the batch of messages.

void C8AdapterSendMessage(C8Adapter *adapter_ptr, C8Message *msg_ptr);

Purpose: sends the specified message to Coral8 Server. 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

After you send a particular msg_ptr, you may not "re-use" that msg_ptr; i.e. you may not put new values into it and send it again. You must 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.

void C8AdapterFlushMessages(C8Adapter *adapter_ptr);

Purpose: force all queued messages to be sent to the server.

The C8AdapterSendMessage() call does not send messages right away. Sent messages are actually sent at the end of the "execute" callback 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 Coral8 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.

After you send a particular message pointer, you may not "re-use" that message pointer; i.e. you may 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.

const C8UserCredentials* C8AdapterCopyCredentials(C8Adapter* adapter);

Purpose: creates a copy of the user 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: the "full" adapter name, e.g. "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" (non-accelerated).

Parameters:

- adapter_ptr - a pointer to the C8Adapter object that stores information about this instance of the adapter.

Returns: returns the accelerated playback rate. A rate of 1.0 is "normal" (non-accelerated).

C8Bool C8AdapterIsPersistent (C8Adapter *adapter_ptr);

Purpose: This returns true if 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: This returns true if 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: nothiing.

void C8AdapterSleep(C8Adapter *adapter_ptr, C8Interval us_delay);

Purpose: The adapter thread sleeps for the specified amount. This is not normally needed because the Coral8 Adapter framework provides appropriate synchronization. This function is provided for any special needs.

The header file c8types.h contains definitions that may be convenient for specifying durations; e.g., to delay 3 seconds, use 3*C8PerSecond.

Note that 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 will almost certainly not be 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: This passes the server a parameter that tells the code that calls the adapter's execute() function how long to wait between calls to the execute() function. (The default is 1 millisecond (1000 microseconds).)

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.

Note that this sleep interval is the time that the CALLER sleeps, not the time that the execute() function sleeps.

In most cases, the default value is best; this function is provided for any special needs.

The header file c8types.h contains definitions that may be convenient for specifying durations; e.g., to delay 3 seconds, use 3*C8PerSecond.

As with any sleep-related function, the actual duration will depend upon how heavily loaded the system is, the granularity of the system clock, etc. The actual sleep will only be an approximation of the amount you request. The actual length of the sleep is likely to be longer than requested if you requested a time smaller than the finest granularity of the system clock, or if the system is heavily loaded. The length of the sleep may be less than the amount of time requested if there is a change in server state (e.g. a shutdown) or if the adapter is an output adapter and data is available for that output adapter. (If CPU time is available, the server will try to call the execute() function as soon as data is available, rather than waiting until the end of the requested sleep length.

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. Note that 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: nothing

void C8AdapterSetSessionState(C8Adapter *adapter_ptr, void *session_state)

Purpose: This stores the "session state", which is a set of user-defined information that the user wants to make available across multiple invocations of the adapter's "execute()" function. Although the session state is retained by the Coral8 engine, 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 crashes and restarts. A session state should thus be created in a user defined struct using `C8Malloc()` or a similar heap-based allocation. In particular, session structures must *not* be defined on the stack as the stack is popped each time the function exits! 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

In the shutdown() callback function, the session state should be set to a Null pointer.

Note that storing and retrieving session information is optional.

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 an 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 will be returned. This indicates no session exists, so the user is responsible for creating a session. The session state is usually constructed by a call to `C8Malloc()` and populating it with a user-defined structure.

Using a static or global for session states is **NOT**thread 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 user; thus the function simply 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 may use the `C8AdapterSetPersistentState()` function to tell the server to make that memory persistent, and you may call this function (`C8AdapterGetPersistentState()`) to retrieve a pointer to that memory. If Studio requested that the stream be persistent, this memory block will persist over reboots and system crashes. The o_data_size pointer permits the use of dynamic structure verification. Depending on the needs of the user, the o_data_size may be ignored for fixed sized structures. The inclusion of o_data_size provides a convenience utility only. Since persistence depends on the size of the user data, a user calling `C8AdapterRealloc()` possibly needs this information. If you have not already called `C8AdapterSetPersistentState()`, then a call to `C8AdapterGetPersistentState()` returns a null pointer. See also the `C8AdapterSetPersistentState()` function.

Do *not* free the pointer returned by this 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 user; thus the function simply returns a pointer to the memory without "interpreting" that memory in any way.

void C8AdapterSetPersistentState(C8Adapter *adapter_ptr, const void* data, C8UInt data_size);

Purpose: This function allows the user to specify that a particular block of memory be made persistent - i.e. 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 may vary from call to call.

Note that only one piece of user memory may 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.

Note that the memory you persist is normally memory that you yourself have allocated (e.g. via `C8Malloc()`) and that therefore you yourself must deallocate (e.g. via `C8Free()`) when you are done with it (e.g. when the adapter's shutdown() function is called). If this memory contains any structures that must be cleaned up (e.g. pointers to other memory that must be deallocated, file handles to files that must be closed, etc.), then make sure that you clean up those 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 that should 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(...);
...
}
```

Again, remember that if the allocated memory itself contains pointers, handles, etc. then those must be cleaned up, too.

Server API

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 Coral8 Server starts, it reads configuration information from an XML file. (By default, this file is named coral8-server.conf.) You may read the configuration information (as of the time that the server started) by using the C8ServerGetPreference() function call.

Parameters:

- i_section - specifies a particular "section" within the configuration file, which is an XML file.
- i_preference - specifies the name of a particular parameter within 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 out the adapters base folder for input and output files used by the adapters supplied by Coral8. The XML for this part of the server configuration file looks like:

```
   ...
<section name="Coral8/Adapters">
   ...
   <section name="ReadWriteBaseFolder">
      <preference name="BaseFolder" value="/c8test/D5"/>
   </section>
</section>
```

Thus the value of the section should be

`"Coral8/Adapters/ReadWriteBaseFolder"`

(Note that since one section was nested within another, we composed a single name that included the entire XML "path", i.e. the whole chain of section names down to the preference.)

And of course the value of the preference should be

`"BaseFolder"`

Sample C code is below:

```
C8Char i_section[] = "Coral8/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 information necessary to describe connections and to perform the adapter function across multiple invocations of the execute() function. Connection-related information includes filenames, stream URIs, etc. Additional persistent state information may include such items as file offsets, intermediate calculations that span multiple rows, processing subtotals or summaries, etc. 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 (e.g., FILE*), transient socket connection information, etc. A session state should contain *only* these 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 will be the same, and the line number at which to resume reading will be the same, so those two pieces of information should be stored in the persistent state. The adapter will have to re-open the data file (e.g. by calling the fopen() function) and may get a different file pointer than it had previously. Thus the file pointer should be stored in the session state rather than in the persistent state. After a crash, you will use the same persistent state information, but will re-generate session state information.

So how does a user determine when the system has crashed and needs to re-initialize? Notice in the execute() function the suggested sequence at the very beginning. Please refer to the "In-process Input Adapter" and "In-process Output Adapter" sections for more details.

```
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 Coral8 engine has been restarted for some reason.
      // Need to recreate the session data.
      if (! my_input_c8adapter_session(i_adapter_ptr)) {
   ...
```

Suggested Session and Persistent State Initialization

The session state should be set up in the initialization routine. If the Coral8 engine or the system crashes, the session state will return a null pointer when C8AdapterGetSessionState() is called. Users 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, etc. Similarly, when the Coral8 engine restarts, the session state is again missing and the same initialization sequence must be performed. The only difference between an initial Coral8 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, the execute() function uses that session state and also the persistent state. The first call to C8AdapterGetPersistentState() in the initialization function will return a null pointer. The user must create whatever memory structure is necessary to perform proper adapter functions. Thereafter, 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. These entry points are identical for both input and output adapters, even though the internal logic will differ. Refer to "In-process Input Adapters", "In-process Output Adapters", 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 (described below). 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_initialize(C8Adapter *i_adapter_ptr)

Purpose: This initialization function prepares the adapter for operation.

The user may perform file opening, socket connections, etc. in anticipation of adapter execution. Depending upon 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, the user is expected to issue appropriate error messages, release any allocated Coral8 resources, and return C8_FALSE. If the Coral8 engine receives a C8_FALSE, the adapter will be considered inactive and no further communications will be attempted.

C8Bool my_input_c8adapter_execute(C8Adapter *i_adapter_ptr)

Purpose: The Coral8 engine will periodically call the user-defined adapter for execution. The user must write the internals of the execute() function according to his own needs. An input adapter differs from an output adapter; please refer to the appropriate sections. The user-defined 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, the user is expected to issue appropriate error messages and return C8_FALSE. The Coral8 engine will then 1) Call the shutdown function, 2) call the initialization function, 3) 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, etc.

void my_input_c8adapter_shutdown(C8Adapter *i_adapter_ptr);

Purpose: The Coral8 engine has determined the adapter must be shut down. Either the associated stream is being shut down by the engine (perhaps directed by Studio), or the execute function needs reinitializing. The user should perform whatever is necessary to close the adapter. This may involve such closing files, sockets, database connections, etc. If the Coral8 engine is closing the stream, the thread of the adapter is being 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 Coral8 engine has determined that the adapter is attempting to perform a Guaranteed Delivery function but is currently disconnected. Your adapter should reconnect with Coral8 Server with C8AdapterConnect. For an input adapter, request the last batch ID. For an output adapter, specify the last batch ID. 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.

In-process Input Adapters

An in-process input adapter gets data from some source and sends the data to the Coral8 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, print schemas, etc. These functions demonstrate the various API calls.

As we described previously, an in-process input adapter must have at least three subroutines:

Initialization: The user must initialize the adapter. This may include such activities as opening sockets, verifying the existence of files and/or directories, pinging network connections, etc. Both persistent and session data do not yet exist and 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: The user queries for any input to send to the Coral8 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 the user's expected behavior of the adapter. If no input is available, the routine should call C8AdapterSleep() and specify a short time period, e.g. one second. (Alternatively, the function may simply return and wait until it is called again.)

Shutdown: The Coral8 engine has received information that the user's 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, etc. Finally, the user must release any resources obtained from the Coral8 engine. In particular, the user must call C8AdapterSetSessionState(adapter_ptr, NULL) to allow the Coral8 engine to release internal resources.

An outline of the user code for an input adapter is shown below.

```
// 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 Coral8 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 Coral8
   ...
   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

A typical in-process output adapter gets data from the Coral8 engine and publishes to some destination. The tutorial output adapter, like the input tutorial adapter, provides additional routines of interest to the adapter developer.

The three callbacks for an output in-process adapter are:

Initialization: The user must initialize the adapter in anticipation to receiving messages form the Coral8 engine and sending the messages to some external data sink. This may include such activities as opening sockets, verifying the existence of files and/or directories, pinging network connections, etc. Any relevant state information created must be saved.

Execution: The user queries for any messages. If a message is available, the user extracts relevant information from the message and handles the information according to the user desired behavior of the adapter. This may include, but is not limited to, creating CSV, XML, ... files, filtering according to some criteria, redirecting to some data sink, ... The exact mechanism is highly dependent upon user intentions. Whether one or multiple messages are processed depends on the Coral8 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. Users may be interested in maintaining connections, querying external events...

Shutdown: The Coral8 engine has received information that the user's adapter should shut down. This may be 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, etc. Finally, the user must release any resources obtained from the Coral8 engine. In particular, the user must call C8AdapterSetSessionState(adapter_ptr, NULL) to allow the Coral8 engine to release internal resources.

A typical pattern for an output adapter would be:

```
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 Adapter

In this section we provide more information about the initialize(), execute(), and shutdown() routines required in an in-process adapter.

Initialization

Our tutorial ADL file contains at least one of each Coral8 data type. Each of these values may be entered in a Properties View (in 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 Coral8 Studio. This printing of parameter values may be observed by displaying an instance of the adapter in Studio. Make sure Coral8 Server is properly connected, then press "Start Module". The adapter initialization routine will be called and the parameter printing routines will display the Studio values on Coral8 Server window.

To change the values and have Coral8 Server notice the changes, press the "Stop Module" in the Studio window. Now change the value of the parameters and press "Start Module" once again. The Coral8 Server log will contain the values that were entered into the adapter's Properties View in Coral8 Studio.

This exercise demonstrates proper communications between the Coral8 Studio and the Coral8 Server.

Another useful printout for initialization would the current working directory of the server. Because Studio may execute on a different machine than Coral8 Server, it is sometimes confusing to determine the working directory of Coral8 Server. Printing this directory aids in this problem.

For this simple example, nothing more is required.

A session state may be required if the user needs to initialize variables, open file/sockets/db connects, ... Remember that the user must define a single block of memory to store this data. This means that you cannot use such useful structures as hash maps, linked lists, etc. (While these data structures may be used, the user is responsible for formatting the data structures into and out of a single memory block.)

The initialization should return C8_TRUE to indicate a proper initialization sequence. If, for some reason, the code is not able to properly initialize the adapter, C8_FALSE should be returned. Reasons for improper initialization may include input files not found, permissions problems, connection problems, etc.

A fatal initialization sequence should, of course, provide details to Coral8 Server log file in whatever helpful manner is adequate to resolve the issue.

Returning a C8_TRUE will continue the adapter execution while a C8_FALSE will produce an error condition visible on 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 that it finishes quickly (preferably in less than 1 second).

Execution

For an in-process input adapter, Coral8 Server will call the execute function over and over. The time period between calls cannot be reliably predicted since this depends heavily upon system load, etc. 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 Coral8 message, and send that message to the Coral8 engine. An input adapter may send more than one message per invocation, but should not send an excessive number of messages. What is excessive? A million messages would be extravagant while 100,000 might be a heavy load. As a general rule of thumb, sending more than 100 messages per execute call is probably 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 Coral8 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 Coral8 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 get all the fields nulled. This may be observed in the input Studio stream viewer.

After the Coral8 engine has received the message, the message gets processed by the Coral8 engine. The user's adapter must destroy the message as seen by the adapter. C8MessageDestroy() performs this function. Failure to perform this destroy *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 Coral8 Server that all is well and no fatal conditions occurred. Problems that might arise include socket timeouts, file permissions, broken connections, ..., anything that is not recoverable. These fatal conditions should return C8_FALSE.

If the execution function returns C8_FALSE the following sequence will be performed *one* time in an attempt to recover:

1. The *shutdown* function will be called. This should properly close/terminate the adapter.
2. The *initialization* function will be called. This will be an attempt to re-initialize the adapter.

3. The *execute* function will be called once again. If this call fails, no further attempts at revival will be performed.

Shutdown

No further messages will be sent to the adapter. The shutdown function should prepare to release any and all resources the adapter has acquired. Actions such as buffer flushing, file closing, socket and/or database disconnection should be performed. Users might wish to print an activity summary on Coral8 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 Coral8 engine. Then the number of lines processed is displayed on the Coral8 window.

The `C8AdapterSetSessionState()` call releases the Coral8 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.

Useful Utility Functions

Below are some "utility" functions that you may find useful.

Printing the Schema

Sending or receiving data with the Coral8 engine requires a schema for the message containing the data. Coral8 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.

Users should 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 get retrieved with `C8SchemaGetColumnName()` and the Coral8 datatypes with another utility function `C8SchemaGetColumnType()`. The output of this function will be displayed on the same window as Coral8 Server since adapters run tightly bound to Coral8 Server.

A minor utility function, `MapC8TypeToString()` converts a Coral8 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

Coral8 Studio provides a Properties View for each instance of an adapter. Depending upon needs, each instance of a user defined adapter may have different run-time parameters. There may be multiple instances of each user adapters present. Each instance of a user defined in-process adapter will be 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 incorrect data type results in an error message on Coral8 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 may choose not to enter these values on the adapter's Properties View in Coral8 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 exactly the same process with different GetParam() data type function calls.

Requirements for the C/C++ File

Your source code will need to #include the proper headers.

```
// Include the Coral8 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, "reconenct()" functions. You may choose any names that you want 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 (not the C++ naming conventions, which are sometimes referred to as "name mangling). To do that, 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);
```

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.

You use one of the examples that Coral8 ships as a starting point. You can find the examples under the Coral8 Repository at **examples/FeatureExamples/Adapters/InProcess**.

Step-by-Step Instructions for Creating an In-process Adapter

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 into a dynamically linkable library (.dll on Microsoft Windows, typically .so on UNIX-like operating systems).
4. Create the .ADL file that describes the adapter's parameters, and which also 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\Coral8\server\bin
   ```

   On UNIX-like operating systems, this is typically

   ```
   /home/<userid>/coral8/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\Coral8\Studio\plugins
   ```

   On UNIX-like operating systems, this is typically

   ```
   /home/<userid>/Coral8/Studio/plugins
   ```

8. Stop both Studio and the server; then re-start both the server and Studio.

Compiling 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 .DLL file.
2. Your list of "include" directories should include 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, then you may use Microsoft's Visual Studio. If you are using 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. 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'd like to use for your project.
   E. Browse and specify the directory in which you'd like 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 will create a simple .cpp file to use as a starting point. We recommend that you remove all the contents of this file and then insert your own C code for the in-process adapter. Make sure that your code contains the #includes and forward declarations described in the section titled Requirements for the C/C++ File.

4. If you have other C-language source files that you need, add them to the project

5. 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 "MySample 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 c8adapters.h file (which is included with the Coral8 product).

      On 32-bit Microsoft Windows, this directory is typically:

      ```
      C:\Program Files\Coral8\Server\sdk\c\include
      ```

      On 64-bit Microsoft Windows, this directory is typically:

      ```
      C:\Program Files (x86)\Coral8\Server\sdk\c\include
      ```

      You may also 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 "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 Coral8 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 Coral8 library.

      On 32-bit Microsoft Windows, this directory is typically:

      ```
      C:\Program Files\Coral8\Server\sdk\c\lib
      ```

On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\Coral8\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. (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". The command line should look similar to the following:

```
/Od /I "C:\Program Files\Coral8\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 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 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\E2\C_SDK\Adapter4\Adapter4\Debug\Adapter4.dll"
  /INCREMENTAL /NOLOGO /LIBPATH:"C:\Program
Files\Coral8\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.

6. To compile, use the appropriate option on the "Build" menu, for example, "Build MySample".
7. Copy the DLL file to the server's "bin" directory.

If you are using a command-line compiler (such as "cc"), please perform the following steps.

1. 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${HOME}/coral8/server/sdk/c/include  \
         -o libmma.so  \
         -L${HOME}/coral8/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'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 (i.e. the 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 Coral8 library to link with to use the Coral8 functions, such as C8GetFloat()

   -fPIC specifies that the compiler should generate Position-Independent Code, i.e. code for dynamic linking.

   -shared specifies that the output will be a shared object library file (as opposed to, for example, a stand-alone executable program).

   WARNING: Although on UNIX-like operating systems your library file name will typically be of the form "libXYZ.so" (e.g. "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" will make platform-specific adjustments for the filename extension (.so vs. .dll) and, if necessary, an initial "lib" prefix.

2. Copy the compiled shared library file to the server's "bin" directory.
3. Copy the .adl file to both the server's plugins directory and Studio's plugins directory.

You may, of course, 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 Studio provides through the C8AdapterGetParamInt() type calls. This provides assurance of proper communication and updating with 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. In developing and deploying a system, the correct working directory for Coral8 Server can sometimes be confusing.

Since your in-process adapter is running as part of the server, if your code "asserts", you will stop the server as well as your adapter. Although you may want to use asserts during the development and debugging states, you should avoid using them in production. Note that some of the example code provided by Coral8 uses asserts. You should remove these asserts (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 having written the C/C++ code for the in-process adapter. Simply put the ADL file in Studio's plugins directory and use it in 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.

Since Coral8 Server and Studio only read ADL files and library files when they start, make sure that you restart the server and Studio any time that you copy an updated ADL file to the plugins directory or copy an updated library file to the bin directory.

All parameters known to 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. Users experimenting with exceptionally large or small values may be encountering overflow/underflow problems.

If there are problems with obtaining typed parameters from Studio, parameters of any data type may be retrieved as a string with C8AdapterGetParamString(). *Remember to C8Free() the returned pointer!* This could be useful for specialized user data types that do not conform to Coral8 data types. A user could display the data type as a string, retrieve with C8AdapterGetParamString() and convert with a specialized conversion routine.

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 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, of course). These directories may be different for release and debug versions. Another way is to select the "Run with clean slate" from the 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 as a normal course. If the stream is persisted and not removed, schema changes or normal starting/stopping can display erratic behavior. Removing the "storage" directory should require re-creation of the workspace as well. This is normal and to be expected.

In the development cycle, adapters will probably have to be modified. Because Studio and Coral8 Server cache adapter information, stop both Studio and the server. Remove the storage directory. Copy the adapter's ADL file to the adapter directories. Now restart Studio. Go to the display of the adapter and remove it from the stream. Reattach the adapter to the stream. If display information was modified, the updated adapter should reflect the update. If there was an ADL coding error such as misspelled XML elements, etc. Studio will refuse to load the adapter and an error message will appear at Studio startup.

C8AdapterGetParamString() can be used to differentiate between a zero value and the empty string. If the parameter field was an integer, and C8AdapterGetParamInt() were called, there is no difference between a 0 value and an empty field. By calling C8AdapterGetParamString(), the adapter author may detect the difference.

To avoid generating overwhelming amounts of data during early testing phases, consider calling the C8Sleep() function between each row that you send. When testing an output adapter, use an input adapter that is configured to send data at a slow rate.

A debugger may be used on user-developed code in the usual manner. The libraries distributed by Coral8 do not contain debugging information.

Performance Optimizations

This section describes some ways to optimize performance of your adapter.

- In most cases, in-process adapters should process multiple rows (if available) each time that the execute() function is called.

This can dramatically improve performance. In most situations, 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 will, of course, be called no more than 1000 times per second. If the adapter processes only 1 row per call, then the adapter will 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.

Of course, the adapter should not process so many rows that it prevents the server from having enough CPU time, either.

You may need to experiment to find the optimal number of rows to process per call, depending upon your hardware, the other processes (besides the server and adapter(s)) that are running on your computer, etc. 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 may 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 for a description of the function. See In-process Output Adapters for sample code that uses this function.

- Change the sleep interval used by the thread that calls the execute() function.

In most cases, you will 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. 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. This sleep value is used by the framework between calls to the execute function. The default sleep value is 1 millisecond.

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 Coral8 Servers (more precisely, across multiple containers working under the same manager). In this situation, you can use Coral8's distributed (parallel) query feature and write a single adapter that will write to multiple input streams.

Similarly, it is possible to write a single adapter that will read from multiple output streams, for example to "merge" data from multiple streams. (Note that in most cases the easiest way to merge streams is through CCL, rather than writing a custom adapter.)

Generally, if you want to read from or write to multiple streams, you will 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 (or read from) 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\Coral8\Server\sdk\c\examples\c8_multistream_adapter.cpp
```

or

```
/home/<userid>/coral8/server/sdk/c/examples/c8_multistream_adapter.cpp
```

Note that this example uses C++ features and therefore cannot be compiled as a pure C program.

Note also that the term "managed" adapter in the source code means the same thing as "in-process" adapter.

Although the adapter uses a combination of in-process and out-of-process API calls, the adapter is considered to be 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`.
- Similarly, 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

If you see a compiler/linker error message similar to the following:

```
error ... second C linkage of overloaded function
'C8ClientSDKInitialize' not allowed
```

then you probably tried 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 Coral8 C/C++ SDK includes a way to register a query.

Before reading this section, please read Creating Streams and CCL Statements from Inside A Program if you have not already done so.

Remember that "registering" a query includes:

- creating the streams specified in the registered query;
- creating the statements specified in the registered query;
- compiling the CCL statements (when we refer to "registering a query", the word "query" refers to zero or more CCL statements)
- loading the query into the server
- connecting ("binding") the stream names in the CCL statements to the existing streams (i.e. to the URIs of those 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).

The project whose streams the registered query binds must be up and running for the `C8RegisterQuery()` function to succeed.

Naturally, if databases or user-defined extensions are used in the queries, the databases and the extensions must be configured in the server (and in the compiler) for the query to work properly.

The query may 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 may also create new input streams and output streams.

The API

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: The purpose of this function is to register a query.

Parameters:

- i_server_uri: the URL of the Coral8 Server (e.g. http://localhost:6789) that will manage the query.
- i_workspace: the name of the workspace in which the query will be 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 (e.g. "$XYZ" in a CCL statement), you may 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 that 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. Please, see the example of how to iterate and free the returned array. Note, that the array may be filled (and therefore will need to be freed) even if the function does not return success.

Returns: The function returns C8_OK or C8_FAIL.

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: 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.

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: name as used in the CCL query text.
- i_schema: stream schema (see schema manipulation methods on how to obtain an instance of a 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: stream name as used in the CCL query text.
- i_schema: stream schema (see schema manipulation methods on how to obtain an instance of a 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: 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 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: destroys (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: parameter name.
- i_value: 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: parameter name.

- i_value: 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: parameter name.
- i_value: 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 boolean type.

Parameters:

- i_name: parameter name.
- i_value: 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 long type.

Parameters:

- i_name: parameter name.
- i_value: 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: parameter name.
- i_value: 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 timestamp type.

Parameters:

- i_name: parameter name.
- i_value: 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: destroys (deallocates) a C8ParameterInfo object.

Parameters:

- i_prm_info: the C8ParameterInfo object to destroy.

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 will contain the streams that our dynamic query will bind to.
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 C8ParameterInfo object to store information about the parameter ($VolumeThreshold) that is referenced in the registered query's CCL statement(s).
6. Register our query.
7. Publish some messages/rows.
8. Read those messages/rows.
9. Stop the registered query.

10. Stop the project whose streams we bound to.
11. Clean up.

The example code below is a modified subset of the example `example_register_query.c` provided with the SDK.

On Microsoft Windows, if you installed to the default directory, the file(s) will be in:

`C:\Program Files\coral8\server\sdk\c\examples`

On UNIX-like operating systems, if you installed to the default directory, the file(s) will be in:

`/home/<userid>/coral8/server/sdk/c/examples`

```
/**
 *  Example for registering a query in C
 * Copyright (C) 2006, Coral8, 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 "c8regqry.h"
int
example_register_query_main(int argc, char **argv)
{
    /*
     * This is an example of using the Coral8 C API
     * for registering queries.
     */
    /* Before this example can be run, Coral8 Server
     * must be running, and a workspace named "Default"
     * must exist on it (create the workspace with
     * Coral8 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 C8ParameterInfo 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 Coral8 Programmer's Guide and CCL
     * Reference 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;
}
```

Again, 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\coral8\server\sdk\c\examples`

On UNIX-like operating systems, if you installed to the default directory, the file(s) will be in:

`/home/<userid>/coral8/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 do the following steps:

- add "`.../coral8/server/sdk/c/include/nspr`" or "`...\coral8\server\sdk\c\include\nspr`" (depending upon your operating system) to the list of include directories (where "..." represents the directory in which you installed Coral8).
- add nspr4.lib and plc4.lib to the list of library files (e.g. 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 may also create your own streams.

We will provide more information in a future release; however, the approach is similar to that shown in the Java SDK chapter.

Troubleshooting

Below are error cases you may encounter:

- The project that contains the streams to which we will bind is not running

  The program won't be able to find out 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. Please check that the server is actually running, the server URL you are passing is the correct URL for this 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 errors would be to try compiling the same CCL files with Coral8 Studio.

Control: Compile/Start/Stop/Status

This section describes how to:

- compile a project (query module) -- i.e. how to compile CCL code.
- start/stop a project (query module).

C8ClientSDKInitialize() must be called prior to calling any of these functions.

Compiling a CCL Project

You may write a C/C++ program that calls the Coral8 compiler to compile a Coral8 project (a `.ccp` file and the files that it references). Note that if you have just a .ccl file without a `.ccp` file, you may also compile that.

You may 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 may set the same options (e.g. 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 Coral8 Compiler API, a set of functions that allow you to compile a Coral8 project file (.ccp) or schema file.

The primary functions in this API are the C8Compile() function and the C8RemoteCompile() function, which essentially do the same thing as the Coral8 compiler (c8_compiler on UNIX or c8_compiler.exe on Microsoft Windows), i.e. they translate a group of CCL statements into a form that the server can use. You can specify the same options for C8Compile() as you can specify on the command line of the Coral8 compiler. (For a description of the command-line parameters of the c8_compiler program, see Compile a Project or a Schema File.)

The API provides additional functions that you may find useful, especially if you want to specify options for the compiler, such as turning warning messages on or off. This section does not explain all those options; we recommend that you read Compile a Project or a Schema File, especially the first table in that section.

Remember: prior to calling any function from the client SDK (be it adapter-, compiler-, or server management-related), 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 may 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 User-Defined Function.)

The functionality provided by this compiler API is similar in at least the following Coral8 SDKs: this C/C++ SDK, the Java SDK, and the .NET SDK.

The Coral8 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: This function 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, 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 Coral8 documentation.

Note that this function only compiles; 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`.)
- i_output_file - where to write the output (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 Coral8 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.
- i_opts - contains 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: This function 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.

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 Coral8 documentation.

Note that this function only compiles; 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`.) Note that 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 output (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 Coral8 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 - contains the compiler options to use or NULL for defaults.
- 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.
- i_compiler_uri - a string containing the URI of the compiler service. Since the URI is being compiled on a remote server, this string simply contains the URI of that remote server, in the usual form: "http://hostname:port", e.g. "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, i.e. the schema file. By convention, the filename extension is normally `.ccs`.
- i_output_file - the name (optionally including the path) of the input file, i.e. the schema file. By convention, the filename extension is normally `.ccx`.

Returns: C8_OK if successful, C8_FAIL otherwise.

C8CompilerOptions * C8CompilerOptionsCreate();

Purpose: Gets a newly allocated compiler options structure containing default values. (The structure must be de-allocated with C8DestroyCompilerOptions().)

Parameters: none.

Returns: a 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. The parameter i_opts must be a valid pointer to the structure to destroy.

Parameters:

- i_opts - a pointer to a C8CompilerOptions structure.

Returns: nothing.

C8Status C8CompilerOptionsGetRepositoryPath (const C8CompilerOptions * i_opts, const C8Char ** o_res);

Purpose: This gets the path to the Coral8 Repository.

Parameters:

- i_opts - valid pointer to a compiler options structure.
- o_res - is 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: This sets the repository path.

Parameters:

- i_opts - a valid pointer to a compiler options 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 - a valid pointer to a compiler options.
- o_res - the location to write the result to.

Returns: C8_TRUE if debug mode, C8_FALSE otherwise.

C8Status C8CompilerOptionsSetDebugMode(C8CompilerOptions * i_opts, C8Bool value);

Purpose: This specifies whether to compile in debug mode.

Parameters:

- i_opts - a valid pointer to a compiler options 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: This function gets the accelerated playback rate, e.g. 10.0 if the data is accelerated by a factor of 10, or 0.5 if the data is 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 playback rate (i.e. 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 - a valid pointer to a compiler options 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 - a valid pointer to a compiler options structure.
- o_res - a location to write the result to.

Returns: C8_OK on success, C8_FAIL otherwise.

C8Status C8CompilerOptionsSetDeprecationWarningsEnabled (C8CompilerOptions * i_opts,
C8Bool value);

Purpose: this function specifies whether to enable or disable deprecation warnings.

Parameters:

- i_opts - a valid pointer to a compiler options structure.
- value - a 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: this function gets a value that indicates whether implicit conversion warnings are enabled.

Parameters:

- i_opts - a valid pointer to a compiler options structure.
- o_res - a 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 - a valid pointer to a compiler options structure.
- value - a 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: this function gets whether the "filter as soon as possible" optimization is enabled.

Parameters:

- i_opts - a valid pointer to a compiler options structure.
- o_res - a 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 - a 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 - a valid pointer to a compiler options structure.
- o_res - a 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 - a 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 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

```
...
#include "c8client.h"
#include "c8compiler.h"
int main(int argc, char **argv)
{
    /*
     * This is an example of using the Coral8 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.

Additional Sources of Information

An example program example_ccl_compile.c is included in the SDK examples.

On Microsoft Windows, if you installed to the default directory, the file(s) will be in:

```
C:\Program Files\coral8\server\sdk\c\examples
```

On UNIX-like operating systems, if you installed to the default directory, the file(s) will be in:

```
/home/<userid>/coral8/server/sdk/c/examples
```

Start/Stop a Project

C8Status C8StartProgram (cost C8Char *i_mgr_uri, const C8Char *i_ws_name, const C8Char *i_filename);

Purpose: starts a compiled program in Coral8 Server. This does not compile the program; you must have already done the compilation separately.

Parameters:

- i_mgr_uri is the URI of the manager.
- i_ws_name is the name of the workspace.
- i_filename is the name of the compiled file (i.e. the .ccx file).

Returns: C8_OK or C8_FAIL.

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 Coral8 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 is the URI of the manager.
- i_ws_name is the name of the workspace.
- i_filename is the name of the compiled file (i.e. the .ccx file).
- i_credentials - - a pointer to a structure containing the user's credentials (username and password).

Returns: C8_OK or C8_FAIL.

C8Status C8StopProgram (const C8Char *i_mgr_uri, const C8Char *i_ws_name, const C8Char *i_progname);

Purpose: Stops a running program in Coral8 Server.

Parameters:

- i_mgr_uri is the URI of the manager.
- i_ws_name is the name of the workspace.
- i_progname is the name of the program (if you compiled the program yourself, it is the "load name")..

Returns: C8_OK or C8_FAIL.

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 Coral8 Server. This version is used if the User Authentication feature limits the right to start and stop programs.

Parameters:

- i_mgr_uri is the URI of the manager.
- i_ws_name is the name of the workspace.
- i_progname is 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 or C8_FAIL.

C8Status C8GetStreamSchema (const C8Char *i_uri);

Purpose: Gets the schema of the stream.

Parameters:

- i_uri is the URI of the stream.

Returns: C8_OK or C8_FAIL.

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 is the URI of the stream.
- i_credentials - - a pointer to a structure containing the user's credentials (username and password).

Returns: C8_OK or C8_FAIL.

Monitoring Servers and Queries

This section provides some information about monitoring servers and queries.

Status API

This section describes the API for getting status information, including information about a Coral8 Manager server, a workspace, or a project (query module).

C8StatusInfo * C8GetManagerStatusInfo(const C8Char * i_manager_uri);

Purpose: Get 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 needs to be freed with C8StatusInfoDestroy().

C8StatusInfo * C8GetManagerStatusInfoA(const C8Char * i_manager_uri, const C8UserCredentials *i_credentials);

Purpose: Get 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 - - a 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 needs to be freed with C8StatusInfoDestroy().

C8StatusInfo * C8GetWorkspaceStatusInfo(const C8Char * i_manager_uri, const C8Char * i_workspace_name, C8Bool i_need_ccx_info);

Purpose: Get 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 needs to 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: Get 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 - - a 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 needs to 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: Get 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 needs to 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: Get 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 - - a 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 needs to be freed with C8StatusInfoDestroy().

void C8StatusInfoDestroy(C8StatusInfo * i_status);

Purpose: Destroys a status object.

Parameters:

- i_status - valid pointer to a valid 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 - valid pointer to a valid status object.
- o_ts - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_name - a name of a message in an object.
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.y

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_name - a name of a message in an object.
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- i_msg_name - a name of a message in an object.
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.y

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_name - a name of a message in an object.
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_msg - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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: Check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_name - a name of a message in an object.
- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**C8Status C8StatusInfoHasMessageByMessageIndex (const C8StatusInfo * i_status,
const C8Char * i_grp_name, const C8Char * i_obj_name, C8UInt i_msg_ndx,
C8Bool* o_res);**

Purpose: Check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).

- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8StatusInfoHasMessageByObjectIndex (const C8StatusInfo * i_status, const C8Char * i_grp_name, C8UInt i_obj_ndx, const C8Char * i_msg_name, C8Bool* o_res);

Purpose: Check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_name - a name of a message in an object.
- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.y

C8Status C8StatusInfoHasMessageByObjectAndMessageIndex (const C8StatusInfo * i_status,
const C8Char * i_grp_name, C8UInt i_obj_ndx, C8UInt i_msg_ndx, C8Bool* o_res);

Purpose: Check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8StatusInfoHasMessageByGroupIndex (const C8StatusInfo * i_status, C8UInt i_grp_ndx, const C8Char * i_obj_name, const C8Char * i_msg_name, C8Bool* o_res);

Purpose: check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- i_msg_name - a name of a message in an object.

- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8StatusInfoHasMessageByGroupAndMessageIndex (const C8StatusInfo * i_status,
C8UInt i_grp_ndx, const C8Char * i_obj_name, C8UInt i_msg_ndx, C8Bool* o_res);

Purpose: check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_res - location where the result is to be written.

Returns:C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8StatusInfoHasMessageByGroupAndObjectIndex (const C8StatusInfo * i_status,
C8UInt i_grp_ndx, C8UInt i_obj_ndx, const C8Char * i_msg_name, C8Bool* o_res);

Purpose: check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_name - a name of a message in an object.
- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8StatusInfoHasMessageByGroupAndObjectAndMessageIndex(const C8StatusInfo * i_status,
C8UInt i_grp_ndx, C8UInt i_obj_ndx, C8UInt i_msg_ndx, C8Bool* o_res);

Purpose: check whether the status info object contains a message.

Parameters:

- i_status - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).

- o_res - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_name - a name of a message in an object.
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.

- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_name - a name of a message in an object.
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- i_msg_name - a name of a message in an object.
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_name - a name of a message in an object.
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_val - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8StatusInfoGetGroupCount(const C8StatusInfo * i_status, C8UInt * o_cnt);

Purpose: Get the number of groups in the status info object.

Parameters:

- i_status - valid pointer to a valid status object.
- o_cnt - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 valid pointer to a valid status object.
- i_grp_ndx a 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 if anything goes wrong.

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 - valid pointer to a valid 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 if anything goes wrong

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- o_cnt - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- o_name - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- o_name - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 valid pointer to a valid status object.
- i_grp_name the name of a group.
- i_obj_name a name of an object in a group.
- o_cnt location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- o_cnt - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_name - a name of an object in a group.
- o_cnt - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- o_cnt - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_name - a name of an object in a group.
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_name - location where the result is to be written.

Returns C8_OK on success, C8_FAIL if anything goes wrong.:

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 - valid pointer to a valid status object.
- i_grp_name - the name of a group.
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_name - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

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 valid pointer to a valid status object.
- i_grp_ndx a valid index of a group (0-based).
- i_obj_name a name of an object in a group.
- i_msg_ndx a valid index of a message in an object (0-based).
- o_name location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

**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 - valid pointer to a valid status object.
- i_grp_ndx - a valid index of a group (0-based).
- i_obj_ndx - a valid index of a message in an object (0-based).
- i_msg_ndx - a valid index of a message in an object (0-based).
- o_name - location where the result is to be written.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

Tracer Message API

A stream may contain not only normal data messages, but also "tracer" messages, which may be used for purposes such as calculating the time required for a message to move through Coral8 Server (from the input stream to the output stream).

Tracer messages are identified by having a message type of C8_MESSAGE_TRACER. For a brief description of different message types, see Message API.

The Coral8 C/C++ SDK includes functions that allow you to get and set the values of fields in these tracer messages. You may get or set the tracer:

- CreationUri - The value should be a valid Coral8 HTTP Stream URI e.g. `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 (actually the 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: Get 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 by the user 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 to which the function should write a pointer to the result.

Returns: C8_OK on success, C8_FAIL if anything goes wrong.

C8Status C8TracerGetCreationTime(const C8Message* i_msg, C8Timestamp* o_res);

Purpose: Get 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 time on success.

Returns: C8_OK on success, C8_FAIL otherwise.

C8Status C8TracerGetCreationFrequency(const C8Message* i_msg, C8Interval* o_res);

Purpose: Get 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: Set 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: Set the creation frequency 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: Set 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

Coral8 permits users to write their own User-Defined Functions (UDFs) in C or C++ and then call those functions from within CCL code.

For example, the user may 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

Since the server must be able to find the library that contains your User-Defined Function, load the library into memory, call your UDF, and pass appropriate parameters to the UDF, 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 may 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 User-Defined Function(s) 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.)
- 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 User-Defined Function and code to "pack" and "unpack" values that are stored in a "context" parameter. (This is described in more detail in Accessing Parameter Values.)
- 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 User-Defined Function that does the actual work that you want (e.g. to return pi), 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 may call the UDF with zero or more parameters. Those parameters may be of any of the data types that CCL supports (INTEGER, FLOAT, STRING, TIMESTAMP, etc.). Although the function is called with (and returns) a CCL data type (INTEGER, FLOAT, STRING, etc.), your C-language UDF must pass (and receive) a C data type (int, double, char *, etc.).

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; i.e. you "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 (i.e. reads the "context" parameter and converts the data from the CCL data types to the C data types);
4. Does the "real work" (e.g. calculates pi, or whatever);
5. "Packs" the return value (i.e. 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 C8U. 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). Note also that in the CCL code, the name is not case-sensitive. You could use "WAVG3", or "wavg3" as well as "WAvg3" when calling the function.

Example UDF

In this example, we'll create a UDF that calculates the weighted average of 3 values. Here is the C signature of our user defined function, along with declarations of the local variables that we will 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);
```

The user must provide an xml implementation of this signature, and must also describe the parameters that will be passed from the CCL statement. The XML would look like:

```
<UserDefinedFunctions
  Name="TestingUserDefinedFunctions"
  Type="ns1:UserDefinedFunctionType"
  xmlns="http://www.coral8.com/udf/2005/04/"
  xmlns:udf="http://www.coral8.com/udf/2005/04/"
  xmlns:ns1="http://www.coral8.com/cpx/2005/04/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
<Description>Coral8 User Function Definition test cases
</Description>
<Vendor>Coral8 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 subdirectory of the server directory and in the plugins subdirectory of the Studio directory. See Compiling a UDF and Putting It in the Correct Directory 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, but not necessarily, the same.
   A. The name of the C function is specified in the line

      Function Name="weightedAverage3"

      and is case-sensitive, just as any C code is case-sensitive.
   B. The name of the function called from CCL is specified in the line

      CclName="WAvg3"

      and is not case-sensitive, just as most CCL code 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 Coral8 loads the library code, it will look for a file with this name and with an extension that is appropriate for the operating system (e.g. .lib, .dll, .so, etc.).
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 was 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)...
```

Note that we used the CCL name (WAvg3), not the C name (weightedAverage3). Note also that in the CCL code, the name is not case-sensitive. Your CCL statement could use "WAVG3", or "wavg3" as well as "WAvg3" when calling the function.

Sample C code containing the UDF is below. A file with this code is included in the installation, under

```
Coral8Repository/version/examples/FeatureExamples/FunctionsAndOperat
ors/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;
}
```

The user may, of course, write any C function at all and may call any 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 our examples, we used a single function to do the real work (e.g. 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 Coral8 Server calls the intermediate function, and the intermediate function calls your UDF. Since the Coral8 Server actually calls the intermediate function, the XML file that describes the UDF must have the name of the intermediate function.

User-Defined Aggregate Functions

For an aggregate function, the user must have a place to store her data in between invocations of the function. For example, if you write your own "SUM()" function, you must store the previous 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 or global variables to store values such as subtotals.

Thus the user's aggregate function must work with the server to store data in a place that will persist after the UDF returns from its call. Coral8 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" (e.g. the window created by a "KEEP 3 ROWS" or "KEEP 10 MINUTES" clause). 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;
...
```

Please note the following:

- 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 (e.g. init_data) to persistent memory that the server allocates. The storage allocated locally (e.g. 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. The user must 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 .udf file (the XML file that describes the function).

You may want to look at the sample code for the runningAverage() function, located in the Coral8 Repository, under **examples/FeatureExamples/FunctionsAndOperators/UserDefinedFunctions/src**.

UDFs: XML Signatures

Coral8 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 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.coral8.com/udf/2005/04/"
    xmlns:udf="http://www.coral8.com/udf/2005/04/"
    xmlns:ns1="http://www.coral8.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 Coral8 user defined function signatures is straightforward.

For the example above:

```
void weightedAverage3(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);
```

The user-defined xml is similar to the following:

```
  <Functions>
   <Function Name="weightedAvg3"
          InitFunctionName="weightedAvg3Init"
          ShutdownFunctionName="weightedAvg3Shutdown"
          Library="user_function_lib"
          CclName="WAvg3"
          IsAggregator="false">
      <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 Name="result" Type="C8Float"/>
    </Output>
 </Function>
</Functions>
```

For each function described in the .udf file, the initial function element is always named "Function" and has the following attributes:

- The "Name" attribute of Function 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.
- The optional "InitFunctionName" attribute 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 (C8SetSessionState) for calls to the function identified with the Name attribute.
- The optional "ShutdownFunctionName" attribute 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 (C8SetSessionState) in the shutdown function.
- The "Library" should be the name of the .dll containing the user function. For UNIX-like operating systems, this will be a .so library. Note that on UNIX-like operating systems, you should not preface the library name 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 (e.g. "select FOO() ... from ..."). The CclName is case insensitive.
- The IsAggregator attribute should be true for aggregate functions and false for other functions.

The "Functions" element of the XML file may describe an arbitrary number of functions. This convenience allows grouping of related functions and/or library modules. 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="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. The "Name" attribute is used for error reporting if there is a type mismatch for any parameters at runtime. These names are passed to the user for similar uses.

The Coral8 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'd names (e.g. "C8Float") rather than the underlying C types (e.g. "double").

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 user defined function.

UDFs: Interface Code

The function will have the following signature:

```
void weightedAverage3(C8Udf *ctx);
```

The user 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 and input/output values, NULL indicators, and meta-information about the function call. The user may choose to ignore meta-information as this contains only supporting information such as parameter names and types. The user must, however, extract the parameters from the input parameter list. Depending upon the application, the user should test each parameter for the NULL status. If a parameter is NULL, the parameter value is undefined.

Coral8 passes data by value as in C. Input values should not be modified or freed; they are read-only values. Memory allocated by Coral8 must be freed by Coral8. Memory allocated by the user must be allocated by the user.

After the user routine finishes, the output value should either contain a value or be set to NULL.

Metadata

const C8Char *C8GetFunctionName(C8Udf *ctx)

Purpose: returns the name of the function as defined in the UDF XML file.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.

Returns: returns the name of the function as defined in the UDF XML file. The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not 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: returns the name of the library as defined in the UDF XML file. The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not 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, etc.

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: given a context, ctx, and a parameter index, provide 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 of the parameters in the context object you want to get the type string for.

Returns: the type string for that parameter as provided in the Parameter attribute in the UDF XML file. The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not C8Free() this pointer.

const C8_TYPES C8GetInputEnumType(C8Udf *ctx, C8UInt ndx)

Purpose: Given a context, ctx, and a parameter index, provide 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. 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 of the parameters in the context object you want 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: given a context, ctx, and a parameter index, provide the name of 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 of the parameters in the context object you want to get the type string for.

Returns: the name of that parameter as provided in the Parameter attribute in the UDF XML file. The user should not change the memory that the pointer points to, and the user should not C8Free() this pointer.

const C8Char *C8GetOutputType(C8Udf *ctx)

Purpose: Given a context, ctx, provide the type string 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: provide the type string for the output parameter. The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not C8Free() this pointer.

const enum C8_TYPES C8GetOutputEnumType(C8Udf *ctx)

Purpose: Given a context, ctx, and a parameter index, provide 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: given a context, ctx, provide the string for the output parameter name. This may be used in error messages or type checking. If the user has not provided an output name in the UDF XML, then this string will be "" (i.e. the empty string).

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.

Returns: the string for the output parameter name. If the user has not provided an output name in the UDF XML, then this string will be "" (i.e. the empty string). The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not C8Free() this pointer.

void C8SetSessionState(C8Udf *ctx, void *data)

Purpose: This stores the "session state", which is a set of user-defined information that the user wants to make available across multiple invocations of the UDF's "execute()" function. Although the session state is retained by the Coral8 engine, 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. 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. In particular, session structures must *not* be defined on the stack as 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

In the shutdown() callback function, the session state should be destroyed.

Note that 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 the user 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: given an pointer to a C8Udf object, return the "session state" information associated with that instance of the UDF. On the initial call to C8GetSessionState(), a null pointer is returned. This indicates no session exists, so the user is responsible for creating a session. The session state is usually constructed by a call to `C8Malloc()` and populating it 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 the user; thus the function simply returns a pointer to the memory without "interpreting" that memory in any way.

Accessing Parameter Values

C8Float C8GetFloat(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, provide the floating point value of that parameter.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter to get the value of. The first parameter is parameter 0.

Returns: the floating point value of that parameter.

C8Int C8GetInt(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, provide the integer value of that parameter.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter 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: given a context, ctx, and a parameter index, provide the long value of that parameter.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter to get the value of. The first parameter is parameter 0.

Returns: the long value of that parameter.

const C8Char *C8GetCharPtr(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, provide the C8Char pointer value of that 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, i.e. a sequence of ASCII characters terminated with a 0 ('\0').

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter to get the value of. The first parameter is parameter 0.

Returns: the C8Char pointer value of that 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, i.e. 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 (e.g. to point to something else).

C8Timestamp C8GetTimestamp(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, provide the timestamp value of that parameter. Coral8 timestamps are in microsecond units.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter to get the value of. The first parameter is parameter 0.

Returns: the timestamp value of the specified parameter. Coral8 timestamps are in microsecond units.

C8Interval C8GetInterval(C8Udf *ctx, C8UInt ndx)

Purpose: Given a context, ctx, and a parameter index, provide the interval value of that parameter. Coral8 intervals are in microsecond units.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter to get the value of. The first parameter is parameter 0.

Returns: the interval value of that parameter. Coral8 intervals are in microsecond units.

C8Bool C8GetBool(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, provide the boolean value of that parameter. Valid C8Boolean values (C8_TRUE and C8_FALSE) are #defined in the c8udf.h file.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter to get the value of. The first parameter is parameter 0.

Returns: the boolean value of that parameter. Valid C8Boolean values (C8_TRUE and C8_FALSE) are #defined in the c8udf.h file.

C8Bool C8GetIsNull(C8Udf *ctx, C8UInt ndx)

Purpose: Given a context, ctx, and a parameter index, return the NULL status for that ndx'th 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 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: given a context, ctx, and a parameter index, return a pointer to the BLOB value of that parameter. If the BLOB is NULL, the function will return 0. If the BLOB is empty, the function will return a non-zero pointer, and the length of the BLOB (returned by the C8GetBlobLength() function) will be 0.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter 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.

⚠ The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not C8Free() this pointer.

C8Char *C8GetBlobBase64String(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, return a pointer to that parameter's BLOB value in the form of a Base64 String. (For information about Base64 String format, see Data Types and Subroutines for UDFs and In-process Adapters.) If the BLOB is NULL, the function will return 0. If the BLOB is empty, the function will return a non-zero pointer, and the length of the BLOB (returned by the C8GetBlobLength() function) will be 0.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- ndx - indicates which parameter 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.)

⚠ The return value is a pointer. The user should not change the memory that the pointer points to, and the user should not C8Free() this pointer.

C8UInt C8GetBlobLength(C8Udf *ctx, C8UInt ndx)

Purpose: given a context, ctx, and a parameter index, return the length of the BLOB (in bytes) of that 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() 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 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: given a context, ctx, and a parameter index, return a pointer to the hexadecimal representation of the BLOB value. The hexadecimal string will consist only of the characters 0..9a..f, with no line breaks or other whitespace. 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 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.)

⚠ The user must C8Free() the returned pointer or a memory leak will occur.

C8Char *C8GetAsStringXml(C8Udf *ctx, C8UInt ndx)

Purpose: Given a context, ctx, and a parameter index, return 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 to get the value of. The first parameter is parameter 0.

Returns: a pointer to the XML field as a string.

⚠ The user must C8Free() the returned pointer or a memory leak will occur.

After the user has evaluated the function, the user should use an output accessor function to return the value to the Coral8 software. The output accessor functions are:

void C8SetOutputInt(C8Udf *ctx, C8Int value)

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.

- value - the value to return.

Returns: nothing.

void C8SetOutputLong(C8Udf *ctx, C8Long value);

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- value - the value to return.

Returns: nothing.

void C8SetOutputFloat(C8Udf *ctx, C8Int value);

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- value - the value to return.

Returns: nothing.

void C8SetOutputTimestamp(C8Udf *ctx, C8Timestamp value);

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- value - the value to return.

Returns: nothing.

void C8SetOutputInterval(C8Udf *ctx, C8Interval value);

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- value - the value to return.

Returns: nothing.

void C8SetOutputBool(C8Udf *ctx, C8Bool value)

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.
- value - the value to return.

Returns: nothing.

void C8SetOutputCharPtr(C8Udf *ctx, C8CharPtr value)

Purpose: Given a context ctx and a value, place the value so that Coral8 can access the value.

The second parameter of this function is a STRING (C8CharPtr). Coral8 will copy the string to its own internal buffer. The user may 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 return.

Returns: nothing.

void C8SetOutputBlob(C8Udf *ctx, const C8Blob *blob, const C8UInt blob_length)

Purpose: Given a context ctx and a BLOB of length blob_length, return 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 Coral8 buffer. (Thus you can and 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.)

- blob_length - the length of the BLOB in bytes.

Returns: nothing.

void C8SetOutputBlobBase64String(C8Udf *ctx, const C8Char *base64_string)

Purpose: Given a context ctx and a BLOB in Base64 String format, return the BLOB as the output value. (For information about Base64 String format, see Data Types and Subroutines for UDFs and In-process Adapters.) The BLOB is copied to an internal Coral8 buffer. (Thus you can and 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.)

Returns: nothing.

void C8SetOutputIsNull(C8Udf *ctx)

Purpose: given a context ctx, the output value is set to NULL. Any other output values will be ignored if this function is used.

Parameters:

- ctx - a "context" object that contains information about the parameters passed to the UDF.

Returns: nothing.

Functions Primarily for Aggregator UDFs

The following functions are used primarily or exclusively in aggregator UDFs:

C8Bool C8IsPositiveMessage(C8Udf *ctx)

Purpose: indicates that the message has entered the window (i.e. is a "positive" message). If so, the user 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. C8_FALSE otherwise.

C8Bool C8IsNegativeMessage(C8Udf *ctx)

Purpose: indicates that the message has exited the window (i.e. is a "negative" message). If so, the user 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: save 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. You may 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: obtain 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. The user should not change the memory that the pointer points to, and the user 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. Some special cautions apply when these are used with in-process operations. See Notes about Allocating and Deallocating Memory in In-process Code.

Compiling a UDF and Putting It in the Correct Directory

The user must code the UDF in C and create a .dll file for windows or a .so file for UNIX-like operating systems. The user must also describe the function signature in terms of the XML file (.udf file) detailed above.

The user must copy the .udf file to both the server and Studio plugins directories, and must copy the .dll (or .so) files to the server bin directory.

On Microsoft Windows, these directories are typically

```
C:\Program Files\Coral8\Server\bin
C:\Program Files\Coral8\Server\plugins
C:\Program Files\Coral8\Studio\plugins
```

On UNIX-like operating systems, these directories are typically

```
/home/<userid>/coral8/server/bin
/home/<userid>/coral8/server/plugins
/home/<userid>/coral8/studio/plugins
```

Since the Coral8 Server and Studio load these files at startup, if the server or Studio is running when the files are copied, the server or Studio must be restarted to load these files.

The user should thoroughly debug the user defined functions in a context other than the Coral8 environment since the Coral8 software contains no debugging information.

Coral8 Access Function Header and Source Files

Coral8 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 the user's own function code. Thus, typedefs for C8Int, etc. as well as access functions are provided. The three context structures are provided as well as code to access these structures. Users must not modify this code as the Coral8 Server and Studio depend upon these structures.

Compiling a UDF

To compile your UDF, you must specify appropriate settings for your compiler, including:

1. Specify that the compiler should generate a shared object file (.so) or a .DLL file.
2. Your list of "include" directories should include the directory that holds c8udf.h.

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 which you'd like 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 will create a simple .cpp file to use as a starting point. We recommend that you remove all the contents of this file and then 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"
```

   Naturally, you will need to customize the names and return types of each UDF (and any intermediate function, if you have one).

4. If you have other C-language source files that you need, add them to the project.
5. 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 "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 file

     `c8udf.h file`

(which is included with the Coral8 product).

On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\Coral8\Server\sdk\c\include
```

On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\Coral8\Server\sdk\c\include
```

You may 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 "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 Coral8 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 Coral8 library.

   On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\Coral8\Server\sdk\c\lib
```

   On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\Coral8\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'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. (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". The command line should look very similar to the following:

```
/Od /I "C:\Program Files\Coral8\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, 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 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\Coral8\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.

6. To compile, use the appropriate option on the "Build" menu, for example, "Build MySample".
7. Copy the DLL file to the server's "bin" directory, and copy the XML (.udf) file to both the server's plugins directory and Studio's plugins directory. On Microsoft Windows, these directories default to:

```
C:\Program Files\Coral8\Server\bin
C:\Program Files\Coral8\Server\plugins
C:\Program Files\Coral8\Studio\plugins
```

If you are using a command line compiler (such as "cc"), you may use a "make" file or a script file. The example below uses a script.

1. Create a script or a "make" file to compile the UDF. 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
CORAL8 = /home/henry/coral8
SDK_PATH = $(CORAL8)/server/sdk/c
INC = -I$(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 $@ $?
```

(Note that you probably will not be able to 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.)

Note that if you are compiling for a 64-bit x86 architecture, you will probably also want to use the `-m64` flag or equivalent.

B. This example uses a script.

```
cc -I${HOME}/coral8/server/sdk/c/include \
     -o libmma.so \
     -L${HOME}/coral8/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 (i.e. the 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 Coral8 library to link with to use the Coral8 functions.

`-fPIC` specifies that the compiler should generate Position-Independent Code, i.e. code for dynamic linking.

`-shared` specifies that the output will be a shared object library file (as opposed to, for example, a stand-alone executable program).

WARNING: Although on UNIX-like operating systems your library file name will typically be of the form "`libXYZ.so`" (e.g. "`libmma.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, an initial "lib" prefix.

Note that if you are compiling for a 64-bit x86 architecture, you will probably also want to use the `-m64` flag or equivalent.

2. Compile the code by executing the script or the "make" file.
3. Copy the shared library file to the server's "bin" directory, e.g.

   ```
   /home/<userid>/coral8/server/bin
   ```

4. Copy the XML (.udf) file to both the server's plugins directory and Studio's plugins directory.

   ```
   /home/<userid>/coral8/server/plugins
   /home/<userid>/coral8/studio/plugins
   ```

UDFs: Summary

For each user-defined function that you want called, you will create two functions:

- The User-Defined Function, which 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 may 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 to the UDF.

UDFs: Cautions

- Because your UDF may be called from multiple queries (and even 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 Coral8 API description specifically states that the Coral8 library function will free the memory).

- Your UDF should not free any memory that is passed to it (unless the Coral8 API description specifically states that the user should free the memory). For example, do not free the context variable that is passed to the user-defined function.
- In CCL statements, your function should be used as a "per-row" function, not as an aggregate function (such as SUM, AVG, etc.)
- You should explicitly handle the following:
    - Error conditions that occur inside your own UDF.
    - NULL input values.
- 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.

Querying a Public Window

Coral8 Server supports "public windows" (created via the CREATE PUBLIC WINDOW statement), which can be queried from outside the server. (For an overview of public windows, see Public Windows. For syntax and other information about the CREATE PUBLIC WINDOW statement, see the *Coral8 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: This 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.

IMPORTANT: 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: This 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 Coral8 Administrator's Guide. For more information about user authentication and the Coral8 C/C++ SDK, see User Authentication.)

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: 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: Gets 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: Gets 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: Get message (representing a row in query result-set) by position.

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.

Important: The message must be deleted with `C8MessageDestroy()`.

```
/* 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

In *Remote Procedure Calls, Database Queries, and Public Windows*, we described Coral8 RPC Plugins, which allow CCL statements to call remote procedures and remote functions. In this section, we will provide more information about compiling and using Coral8 RPC Plugins.

The information in this section is based on the RPC plugins that Coral8 supplies, but the steps should be nearly identical for any RPC plugin, including RPC plugins written by customers or third parties.

Remember, Coral8 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 may use the Coral8-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.

RPC Plugin API

This section lists the functions in the RPC Plugin API. The header file with the function prototypes is c8rpc.h.

Note that for each 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 just 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

For each RPC plugin, the coral8-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 coral8-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

C8Status C8RpcReturnMessage(C8Rpc *rpcctx, C8Message *msgptr);

Purpose: Returns an output message from the plugin. This function can be called multiple times to return multiple messages. 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 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", and the Coral8 C/C++ SDK provides functions that allow you to store and retrieve session state. Note that session state information is not carried over if the server crashes or is restarted. (Note that, 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 the user sets the state (e.g. in the initialize() function or the execute() function), the server stores a copy of the pointer to this memory. When the user calls the `C8RpcGetSessionState()` function (e.g. in the execute() function or the shutdown() function), the server returns that same pointer.

Parameters:

- rpcctx - a pointer to the "context" (data structure) for this particular instance of the plugin.
- state - This is 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" (data structure) 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 Functions 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

The RPC Plugin must be compiled into a .dll file for windows or a .so file for UNIX-like operating systems.

The user must copy the .dll (or .so) files to the server bin directory.

On Microsoft Windows, the directory is typically

```
C:\Program Files\Coral8\Server\bin
```

`On UNIX-like operating systems, the directory is typically`

```
/home/<userid>/coral8/server/bin
```

Since the Coral8 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.

The user should thoroughly debug the user defined functions in a context other than the Coral8 environment since the Coral8 software contains no debugging information.

The user must also update the `coral8-services.xml` file to specify the library and function names. (See the *Coral8 Administrator's Guide* for details about updating the `coral8-services.xml` file.)

Coral8 Source files

Coral8 provides three source code files related to RPC plugins:

- c8_rpc_http.cpp -- This is the Coral8 RPC plugin that uses the HTTP protocol.
- c8_rpc_soap.cpp -- This is the Coral8 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 may use these as-is or you may make a copy and customize it.

Compiling an RPC Plugin

To compile an RPC plugin, you must specify appropriate settings for your compiler, including:

1. Specify that the compiler should generate a shared object file (.so) or a .DLL file.
2. The list of "include" directories should include the directory that holds c8_http_client.h. This could be either the directory in which Coral8 originally supplied this file (`server\sdk\c\examples`) or a separate directory that you have created and to which you have copied the file.

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 which you'd like 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 will create a simple .cpp file to use as a starting point. We recommend that you remove all the contents of this file and then 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 may customize the names shown in bold. However, the names used in the source code file and the names used in the coral8-services.xml file must match.

4. If you have other C-language source files that you need, add them to the project.

5. 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 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
```

```
c8_client_http.h
```

(which are included with the Coral8 product).

On 32-bit Microsoft Windows, this directory containing c8rpc.h is typically:

```
C:\Program Files\Coral8\Server\sdk\c\include
```

On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\Coral8\Server\sdk\c\include
```

The directory containing `c8_http_client.h` depends upon where you've copied this file to.

You may 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 Coral8 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 Coral8 library.

   On 32-bit Microsoft Windows, this directory is typically:

```
C:\Program Files\Coral8\Server\sdk\c\lib
```

   On 64-bit Microsoft Windows, this directory is typically:

```
C:\Program Files (x86)\Coral8\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
```

   (Note that 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 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 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. On Microsoft Windows, this directory defaults to:

   `C:\Program Files\Coral8\Server\bin`

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${HOME}/coral8/server/sdk/c/include \
           -o libmma.so \
           -L${HOME}/coral8/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 (i.e. the 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 Coral8 libraries to link with to use the Coral8 functions.

   `-fPIC` specifies that the compiler should generate Position-Independent Code, i.e. code for dynamic linking.

   `-shared` specifies that the output will be a shared object library file (as opposed to, for example, 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, e.g.

   `/home/<userid>/coral8/server/bin`

User Authentication

In this section, we discuss using the Coral8 C/C++ SDK with user authentication. We assume that you have already read (*User-Defined Functions and Plugins* and *Server Plugins*). If you are not using the User Authentication feature of Coral8 Engine Enterprise Edition, you do not need to read this section.

In the first part of this section, we show the functions in the Coral8 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

If you are using the Enterprise Edition of the Coral8 engine and you have enabled the User Authentication feature, you may need to get and set "credentials". A "credential" is a data structure that contains a user's ID and password. When you want to perform an action that requires authentication, you 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 -- e.g. the function to subscribe to a stream.

(Remember that specifying credentials will add access only to those resources (workspaces, streams, etc.) and actions (create, read, startProgram...) for which the specified user was granted privileges in the ACL file (`coral8-acl.xml`). If the user wasn't granted acces to a particular resource/action, then specifying Credentials will not give you access.)

The following parts of the Coral8 C/C++ API. These functions allow you to get and set "credential" information. The functions that actually "use" these credentials (e.g. to subscribe to a stream) 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 C8UserCredentialsSetPassword(). 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 in which the username should be written.
- i_user - the username

Returns: C8_OK or C8_FAIL.

C8Status C8UserCredentialsSetPassword (C8UserCredentials* i_credentials, const C8Char* i_password);

Purpose: fills in the password field in the user credentials structure.

Parameters:

- i_credentials - a valid pointer to user credentials.
- i_password - 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 (e.g. 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

As described elsewhere (*User-Defined Functions and Plugins* and *Server Plugins*), if the authentication plugins (LDAP and htpasswd) supplied by Coral8 do not meet your needs, you may write your own plugin to authenticate users. This section describes the API for writing your own plugin.

The Coral8 Server 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. The user then passes this pointer on to other library functions when calling them. The data structure that the plugin pointer points to is "opaque"; the user does not need to read it or change.

The authentication context holds information about the user (username, ID, and the names of the groups that this user is a member of) and the information about the authentication results:

```
/**
 * 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: This gets the user name from an existing authorization context.

Parameters:

- ctxt - a pointer to the authorization context from which you wish to retrieve the user name.

Returns: a string containing the username (also called the "username").

const char * C8AuthContextGetpassword (C8AuthContext *ctxt);

Purpose: This gets the user password from an existing authorization context.

Parameters:

- ctxt - a pointer to the authorization context from which you wish to retrieve the user's password.

Returns: a string containing the password.

void C8AuthContextSetResult (C8AuthContext *ctxt, C8Int res);

Purpose: This sets a value that indicates the result of the authentication, i.e. whether the user was authenticated or not. The possible values (C8_AUTH_SUCCESS, C8_AUTH_FAILURE, etc.) are enumerated above.

Parameters:

- ctxt - a pointer to the authorization context for which you wish to set the result.
- res - the result (e.g. C8_AUTH_SUCCESS).

Returns: nothing.

void C8AuthContextAddGroup (C8AuthContext *ctxt, const char *groupName);

Purpose: This adds a group to the list of groups that the user is a member of. Note that a single user may be a member of multiple groups. You may call this function more than once for a single user (i.e. for a single ctxt); each time that you call, the group name that you specify will be added to the list of group names for this user.

Parameters:

- ctxt - a pointer to the authorization context for which you wish to set the result.
- groupName - the name of a group that this user is a member of.

Returns: nothing.

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 Coral8 Server plugin, an authentication plugin contains "initialize()", "execute()" (i.e. "authenticate") and "shutdown()" functions. Below we show some examples with the correct input parameter types and return types. Note that the function names may be different; you specify the actual names of the function in the "AccessControl/Authentication/plugin" section of the server configuration file (this will be explained in more detail later).

C8Status c8authplugin_demo_initialize(C8AuthPlugin *plugptr);

Purpose: This does any initialization that the plugin needs. This may involve allocating memory, opening a file, connecting to an authentication service, etc.

Parameters:

- plugPtr - a pointer to a C8AuthPlugin object.

Returns: C8_OK if successful, C8_FAIL otherwise.

C8Status c8authplugin_demo_authenticate(C8AuthPlugin *plugPtr, C8AuthContext *);

Purpose: This is the function that is called each time that the server needs to authenticate 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: This does any cleanup that the plugin needs. This may involve de-allocating memory, closing a file, etc.

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, such as file pointers, that the user needs each time that the execute/authenticate 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, such as file pointers, that the user needs each time that the execute/authenticate function is called. Note: 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.

Note that 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. Note that you will need to #include the c8auth_plugin.h file

Plugin Configuration

The authentication plugin is declared in the Coral8/Security/AccessControl/Authentication/Plugin section of the Coral8 Server configuration file (`coral8-server.conf`). Below is a sample configuration section. You will customize the actual names of the library file and the 3 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 may be passed to a plugin by specifying a preference with any name in the Coral8/Security/AccessControl/Authentication/Plugin section of the server configuration file (`coral8-server.conf`). For example, in the htpasswd plugin provided by Coral8, 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
Coral8/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);
```

E.g. to read the "MyPreference" preference above, one would call:

```
C8Char *pref = C8AuthPluginGetPreference(plugPtr, "MyPreference",
                                                  "default");
```

Library-Wide Initialization and Shutdown

As we mentioned earlier, one library (.dll or .so file) for Coral8 Server may contain more than one UDF, or more than one in-process adapter, or more than one server plugin. In some cases, you may wish to have initialize() and shutdown() functions that apply to the entire library (in addition to those that apply for a particular in-process adapter, etc.) and that are called once at the time that the library is loaded or unloaded by Coral8 Server.

To implement this, you write 2 functions with the following signatures:

C8Int c8_library_initialize()

Purpose: This function is called immediately after the library is loaded. This function can do any initialization required for the library as a whole. This function can also return a value to the Coral8 system indicating whether the library is unloadable or not.

Parameters: none.

Returns: This function returns to Coral8 Server 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. Furthermore, Coral8 Server may unload the library if there are no remaining references to it (e.g. 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. This function can do any cleanup required for the library as a whole.

Returns: nothing.

These functions are purely optional. In your library, you may include either function, both functions, or neither function.

The function prototypes and the C8LibraryResult enumeration (which defines the return codes such as C8_LIBRARY_INIT_ERROR) are in the c8server.h file in the "include" subdirectory of the Coral8 C/C++ SDK directory. If you installed to the default directory on Microsoft Windows, this will be:

```
C:\Program Files\Coral8\Server\sdk\c\include
```

If you installed to the default directory on a UNIX-like operating system, this will be:

```
/home/user/coral8/server/sdk/c/include
```

Coral8 Java SDK

You use the Coral8 Java SDK to write out-of-process adapters, control Coral8 Engine, register queries, and 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.

Note that Coral8 only validates against the Sun JDK, and not any other Java development environment. See "Coral8 Engine Third-Party Software Dependencies" in the *Coral8 Administrator's Guide* for version information.

Locating Files

When you install Coral8 Server or Coral8 Studio, you also install the Coral8 Java SDK. For installation instructions, see the *Coral8 Administrator's Guide*. You will find all of the Java SDK files in the directory **java5** under the Coral8 Server or Coral8 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.

Coral8 ships several examples with the Java SDK, which you will find under **java5/examples/com/coral8/sdk/examples**. The examples are thoroughly commented to help you understand the purpose of each line of code.

Setting Up Your Environment

The only task you need to perform to set up your development environment for the Coral8 Java SDK is to 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**. If you need to use a different version of any of the files under **java5/lib**, specify each of those separately in your CLASSPATH.

Using the Examples

The Java 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 Java SDK 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, be sure to read the embedded comments. 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 Coral8 compiler (**bin/c8_compiler** under your Coral8 Studio or Coral8 Server installation) by passing the **–D** option to the JVM to specify a value for the variable **c8.compilerPath**.

Examining Example 1: Subscribing to a Stream

If you are writing an output adapter, you will need to subscribe to a stream. Typically, you will 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. For more information about adapters, see Adapters.

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 Coral8 Engine 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

First you create an instance of a client to connect to and communicate with an instance of Coral8 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 Coral8 Server process. The **newEngineClient** method takes parameters specifying the HTTP URL of the host running Coral8 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 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

Now you need to create the CCL URI of the stream you will be getting data from (subscribing to):

```
String cclUri = CclUriFactory.newCclStreamUri(
   serverUrl, CORAL8_JAVA_SDK_EXAMPLES_WORKSPACE,
   "SensorNetwork", "SensorReadings");
```

The parameters passed to this method include the URL of the Coral8 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

Now you're ready to subscribe to the stream:

```
subscription = engineClient.subscribeToStream(cclUri);
```

Here the code is using 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

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 will 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

At the end of processing, the example disconnects the subscription:

```
subscription.disconnect();
```

Compiling and Running

Navigate to the examples directory under the installation: **sdk/java5/examples**.

Enter the following command:

```
javac com/coral8/sdk/examples/Example_01_GettingDataFromAStream.java
```

Run the compiled project by entering the following command, replacing *<PathToCoral8Compiler>* with the full path to and name of the Coral8 compiler on your system, typically **bin/c8_compiler** under the installation directory:

```
java -Dc8.compilerPath=<PathToCoral8Compiler>
  com.coral8.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 (tuples) from the data stream.

Other Examples

The rest of this chapter describes each of the other Java 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

Name	Example_02_PublishingDataToAStream
Description	This example demonstrates the steps necessary to publish data to a stream, a typical activity when writing an input adapter (take data from an outside source and publish it to a stream feeding a query module).
Notable Classes and	**Publisher** engineClient.**createPublisher**(cclUriOfInputStream)

Methods	publisher.**publishRow**(point) publisher.**disconnect**()
Related Information	For more information about streams, see Data Stream. For more information about adapters, see Adapters.

Controlling the Engine

Name	Example_03_ControllingTheCoral8Engine
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**(serverVersion) **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.

Registering 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.

Compiling and Starting a Project

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.

Exploring Value Types

Name	Example_06_ExploringCoral8ValueTypes
Description	This example creates and manipulates objects of various types, demonstrating the data types available in the Java 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

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.

Working with Tuples

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.

Retrieving Server Status

Name	Example_09_QueryingCoral8EngineStatus
Description	This example demonstrates how to retrieve information about an instance of Coral8 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 and Status Information.

Publishing 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 rather 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

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 rather 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.**getAllAvailableTuples**()

Working with Bundles

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.

Guaranteeing Message Delivery

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.

Registering a Query with Parameters

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.

Working with URIs

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

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.

Working with Parallel Queries

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 *Coral8 Administrator's Guide*.

Coral8 .NET SDK

You use the Coral8 .NET SDK to write out-of-process adapters, control Coral8 Engine, register queries, and 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 Files

When you install Coral8 Server or Coral8 Studio, you also install the Coral8 .NET SDK. For installation instructions, see the *Coral8 Administrator's Guide*. You will find all of the .NET SDK files in the directory **net3** under the Coral8 Server or Coral8 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.

Coral8 ships several examples with the .NET SDK. You will find compiled versions of the examples under **net3/examples** and the corresponding source code under **net3/examples/src**. The examples are thoroughly commented to help you understand the purpose of each line of code.

Using the Examples

The .NET 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 .NET 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, be sure to read the embedded comments. 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 Coral8 compiler (**bin/c8_compiler** under your Coral8 Studio or Coral8 Server installation) by setting a value for the variable **c8.compilerPath**.

Examining Example 1

If you are writing an output adapter, you will need to subscribe to a stream. Typically, you will 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. For more information about adapters, see Adapters.

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 Coral8 Engine 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

First you create an instance of a client to connect to and communicate with an instance of Coral8 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 Coral8 Server process. The **NewEngineClient** method takes parameters specifying the HTTP URL of the host running Coral8 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 .NET 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

Now you need to create the CCL URI of the stream you will be getting data from (subscribing to):

```
string cclUri = CclUriFactory.NewCclStreamUri(
   serverUrl, CORAL8_NET_SDK_EXAMPLES_WORKSPACE,
   "SensorNetwork", "SensorReadings");
```

The parameters passed to this method include the URL of the Coral8 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

Now you're ready to subscribe to the stream:

```
subscription = engineClient.SubscribeToStream(cclUri);
```

Here the code is using 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

Now you can read a tuple (row) 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 will 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

At the end of processing, the example disconnects the subscription:

```
subscription.Disconnect();
```

Compiling and Running

You can compile the example source code from command line using the **csc** compiler and referencing the file **Coral8-*x.y.z*.dll**, where *x.y,z* is the Coral8 Engine version number. You can also load the project file **Examples.csproj** into Visual Studio, set up a reference to the file **Coral8-*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

The rest of this chapter describes each of the other .NET 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

Name	Example_02_PublishingDataToAStream
Description	This example demonstrates the steps necessary to publish data to a stream, a typical activity when writing an input adapter (take data from an outside source and publish it to a stream feeding a query module).
Notable Classes and Methods	**IPublisher** engineClient.**CreatePublisher**(cclUriOfInputStream) publisher.**PublishRow**(point) publisher.**Disconnect**()
Related Information	For more information about streams, see Data Stream. For more information about adapters, see Adapters.

Controlling the Engine

Name	Example_03_ControllingTheCoral8Engine
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**(serverVersion) **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.

Registering 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.

Compiling and Starting a Project

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	CclCompilerFactory.NewInstance().**NewLocalCompiler**() localCompiler.**Compile**

Methods	CclCompilerFactory.NewInstance().**NewRemoteCompiler** e.**CompilerOutput**()
Related Information	For more information about compiling and starting projects, see Commands.

Exploring Value Types

Name	Example_06_ExploringCoral8ValueTypes
Description	This example creates and manipulates objects of various types, demonstrating the data types available in the .NET 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

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.

Working with Tuples

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.

Retrieving Server Status

Name	Example_09_QueryingCoral8EngineStatus
Description	This example demonstrates how to retrieve information about an instance of Coral8 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 and Status Information.

Working with Bundles

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.

Guaranteeing Message Delivery

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.

Registering a Query with Parameters

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

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

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.

Working with Parallel Queries

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 *Coral8 Administrator's Guide*.

Coral8 Perl SDK

This chapter describes how to use the Coral8 Perl SDK, which allows you to do things such as write adapters in Perl.

Prerequisites

This chapter assumes that you have already installed Perl and are fluent in the language.

Version of Perl. The Coral8 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. Coral8 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 Coral8 Perl SDK, see Installation and Configuration.

API Interface

As we mentioned earlier, a typical adapter performs the following tasks:

1. Acquire an "address" (a URI) that will uniquely identify a specific stream and tell the communication layer (provided by Coral8) how to find that stream.
2. Open a connection to the stream.
3. Write the desired data. Typically the write operation is in a loop; the adapter will keep sending multiple rows of data.
4. Close the connection.

When you use 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 Coral8 Studio. Right-click on the stream, then choose "properties", and read the stream's URI in the "Stream URI".

When you use 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

The Perl API consists of three parts:

- a tuple API for manipulating Coral8 messages (C8::Tuple module)
- a publisher API for sending data to Coral8 Server (C8::Publisher module)
- a subscriber API for receiving data from Coral8 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 Coral8 Perl SDK, including example source code, is available in the sdk/perl subfolder of your Coral8 Server installation.

Below, we describe all of the functions in the API:

C8::Tuple

A C8::Tuple object represents a tuple - 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 field/column values from the tuple object, and then write the data to whatever destination you want.

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 since midnight January 1, 1970. WARNING: This value is in seconds, not microseconds. NOTE: The Perl library contains functions to convert between seconds and more human-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 since midnight January 1, 1970.

WARNING: This value is in seconds, not microseconds.

NOTE: The Perl library contains functions to convert between seconds and more human-readable formats such as "YYYY-MM-DD HH24:MI:SS".

fields()

Get the tuple fields list. This gets a complete row. The data is returned as an array, with one field per array element.

fields($val)

Set the tuple fields list. This sets a complete row. The data is passed as an array, with one field per array element.

field($name)

This gets a single value from the current tuple. Given the name of the field, return the value as a string.

field($name,$val)

This 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 (microseconds since 00:00:00 Jan 1, 1970).

WARNING: The precision is in seconds, not microseconds.

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 (microseconds since 00:00:00 Jan 1, 1970).

WARNING: The precision is in seconds, not microseconds.

C8::Publisher

You create a C8::Publisher object if you want to write data to a stream. You need one C8::Publisher object per stream.

new()

Creates a new C8::Publisher object.

connection($uri)

Creates a publisher connection to the given $uri that identifies a Coral8 Stream. Returns 1 if the connection is ready to receive data. Returns an undefined value if an error occurs.

write_tuple($tuple)

Writes 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

You create a C8::Subscriber object if you want to read data from a stream. You need one C8::Subscriber object per stream.

new()

Creates a new C8::Subscriber object.

connect($uri)

Creates a subscription connection to the given $uri that identifies a Coral8 Stream.

Returns 1 if the subscription is ready to receive data.

Returns an undefined value if an error occurs.

read_tuple()

Gets the next message from the stream. Note that the first row returned is always the title row (which contains the names of the fields). This is true even if you have connected to a stream that has transmitted messages to other subscribers. (A stream may have more than one subscriber.)

Note that calls to read_tuple() are blocking calls. The call will not return until a message is received on the stream.

Returns the C8::Tuple object or an undefined value if there are no more messages in the stream or a connection error occurs.

column_names()

Gets the list of column names (valid only after 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 Coral8 Stream)

The following example demonstrates a simple input adapter that publishes generated messages to the Coral8 Stream. The Perl program connects to the Coral8 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 Coral8 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 Coral8 Stream URI from the command line.

Lines 8-10. Create Coral8 publisher and connect to the Coral8 Stream with given URI.

Lines 12-13. Declare the list of field names in the tuple.

Lines 15-26. The main loop: create a tuple, publish it to Coral8, sleep; create a tuple, publish it to Coral8, sleep; ...

Lines 16-17. Create a tuple with given list of fields.

Lines 19-20. Set tuple fields to given values.

Lines 22-23. Publish newly created tuple.

Line 25. Sleep a little bit.

Perl Output Adapter (Receiving Data from a Coral8 Stream)

The following example receives data from a Coral8 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. Load the Coral8 module C8::Subscriber. The module must be available in the Perl include path (see Perl documentation for more details).

Lines 3-5. Get the Coral8 Stream URI from the command line.

Lines 7-9. Create Coral8 subscriber and connect to the Coral8 Stream with given URI.

Lines 11-12. Get the list of column names from the Coral8 Server and print it out.

Lines 15-19. The main loop: read a tuple, print it out; read a tuple, print it out; ...

Line 16. Read the new tuple.

Lines 17-18. Print tuple.

Installation and Configuration

This section explains how to run the Coral8 Perl SDK installation script and how to specify the path to the library files.

Running the Installation Script

Before you use the Coral8 Perl SDK, you should run the installation script, named Makefile.PL.

```
perl Makefile.PL
```

On Microsoft Windows, this file is typically in the directory:

```
C:\Program Files\Coral8\Server\sdk\perl\C8
```

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

```
/home/<userID>/coral8/server/sdk/perl/C8
```

If you install modules into the site-specific perl library directories, you will probably 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 do need to install the libraries in such a way that perl will find them. To install into personal directories, you may 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

If you have not already done so, you must download and install the Perl modules that the Coral8 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: we execute a command that goes to CPAN (Comprehensive Perl Archive Network).

Line 2: when prompted, we enter the names of the libraries that we want to download from CPAN.

Line 3: exit the perl interpreter.

Note that these commands ("install ..." and "quit") should be entered when you are prompted to enter commands. Depending upon how your system is initially configured, you may be asked various questions before you are shown the prompts at which you should enter these two commands.

To use the Coral8 Perl API, the directories that contain the Coral8 Perl modules must be in the Perl include path. The Coral8 Perl modules are in the subdirectories:

```
server/sdk/perl/C8
server/sdk/perl/C8/C8
```

under the directory in which you installed the Coral8 product.

On Microsoft Windows, these are typically:

```
C:\Program Files\Coral8\Server\sdk\perl\C8
C:\Program Files\Coral8\Server\sdk\perl\C8\C8
```

On UNIX-like operating systems, these are typically:

```
/home/<userID>/coral8/server/sdk/perl/C8
/home/<userID>/coral8/server/sdk/perl/C8/C8
```

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 Example

To run the example that Coral8 supplies, do the following:

1. Start Coral8 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 Coral8 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).

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>
   ```

Note that 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 the real world, you would also have:

1. created an output stream
2. looked up the URI of that output stream
3. launched the subscribe.pl script with the URL of the output stream rather than the URL of the input stream, and.
4. of course you would also have written one or more queries that fed data into the output stream.

Troubleshooting

This section provides some troubleshooting tips that are specific to the Coral8 Perl SDK.

Additional troubleshooting tips are in Troubleshooting.

You get an error similar to the following:

```
Can't locate C8/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/c8_publish.pl line 10. . . .
```

The most likely cause is that you have not set your perl "include" path to include the Coral8 directory that includes the Coral8 perl modules. To configure your environment properly, see Prerequisites.

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 (Comprehensive Perl Archive Network modules). For a list of these modules, see Specifying the Path to the Library Files. If you have not already acquired these modules, you may need to acquire them. If you have acquired them, you need to set your perl "include" path to include these modules.

Coral8 Python SDK

This chapter provides a brief explanation of the Coral8 Python SDK, which allows you to do the following:

- Write an input adapter or an output adapter in Python.
- Start or stop the Coral8 Server from inside a python program.

API Interface

As we mentioned earlier, a typical adapter performs the following tasks:

1. Acquire an "address" (a URI) that will uniquely identify a specific stream and tell the communication layer (provided by Coral8) how to find that stream.
2. Open a connection to the stream.
3. Write the desired data. Typically the write operation is in a loop; the adapter will keep sending multiple rows of data.
4. Close 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 Coral8 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

The Python API consists of three parts:

- a Tuple class for manipulating Coral8 messages
- a Publisher class for sending data to Coral8 Server
- a Subscriber class for receiving data from Coral8 Server

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 a Publisher object to send the data to the server. Similarly, in an output adapter, you would 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 Coral8 Python SDK, including example source code, is available in the **sdk/python** subfolder of your Coral8 Server installation. The primary files are:

- sdk/python/coral8.py file, which contains the Tuple, Publisher, and Subscriber classes;
- sdk/python/examples/c8-pubsub.py file, which contains example code using a Publisher and a Subscriber;
- sdk/python/examples/c8-server.py file, which contains example code to start and stop Coral8 Server.

Below, we describe all of the functions in the API:

Tuple

A Tuple object represents a tuple - 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 field/column values from the tuple object, and then write the data to whatever destination you want.

getcolumnnames()

Returns a list that contains the names of the tuple columns.

gettimestamp()

Gets the tuple's timestamp, expressed as the number of seconds since midnight January 1, 1970.

WARNING: This value is in seconds, not microseconds.

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 (Comma-Separated Value) string. Note that 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 the row timestamp value (in microseconds since 00:00:00 Jan 1, 1970).

Publisher

You create a Publisher object if you want to write data to a stream. You need one Publisher object per stream.

write_tuple(self, tuple)

Writes a 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.

Subscriber

You create a Subscriber object if you want 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).

Note: 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. Note that the first row returned is always the title row (which contains the names of the fields). This is true even if you have connected to a stream that has transmitted messages to other subscribers. (A stream may have more than one subscriber.)

Note that 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 Coral8 Stream)

The following example demonstrates a simple publisher that writes messages to a Coral8 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 coral8 import Coral8
# When you start this program, pass:
# 1) the URI of the stream to publish to, e.g.
#    "ccl://localhost:6789/Stream/Default/Subscriber2/StreamIn1"
# 2-N) the column names of the stream
class Coral8Test(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 = Coral8.Publisher(self.uri)
   j = 0
   while True:
     tuple = Coral8.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 = Coral8Test('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 Coral8 Stream)

The following reads (subscribes to) data in a Coral8 stream. This code could be extended to make an output adapter.

The listing is below:

```
from coral8 import Coral8
# 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 = Coral8.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 Environment

Make sure that the python interpreter can find the coral8.py file

```
server/sdk/python/coral8.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.

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 Coral8, allows the user to specify which file to read from. Similarly, if you write your own in-process adapter, you may use parameters to allow a user to specify the exact data source to read from (in the case of an input adapter) or the exact data destination to write to (in the case of an output adapter). 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 Coral8 Studio. The Explorer View of Coral8 Studio will display the adapter properties, such as Filename, Loop count, Rate, etc.

ADL allows you to describe your parameters in such a way that Coral8 Studio can display labels and empty fields to the user, and then pass the filled-in values to your in-process adapter. For example, your adapter might use ADL to tell Studio that it should display the label "Filename", read the string that a user types into a field next to that label, and then pass the user's 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
- InstanceParameterDefinitions

Each InstanceParameter Definition comprises one or more Parameter elements that each contain:

- Name

- Type
- Default Value
- Description
- A flag to indicate whether the parameter is required or not (the user may be allowed to leave some parameters blank)

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.coral8.com/adl/2005/04/"
Name="ReadFromXmlFileAdapterType"
xmlns:ns1="http://www.coral8.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>Coral8, 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>
```

NOTE: When Coral8 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 the c8_read_from_csv_file.adl file) 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 Coral8 can be found in the directory

```
<install_dir>/Studio/plugins
```

On Microsoft Windows, this is typically

```
C:\Program Files\coral8\Studio\plugins
```

On UNIX-like operating systems, this is typically

```
/home/username/coral8/studio/plugins
```

The ADL file begins with a section similar to that shown below.

```
<AdapterDefinition
xmlns="http://www.coral8.com/adl/2005/07/"
Name="UserDefinedReadFromCsvFileAdapterType"
xmlns:ns1="http://www.coral8.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>Coral8, 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" may require modification to Direction="Output" for an output adapter.

Specify SupportsGuaranteedDelivery="true" if yours is a Guaranteed Delivery adapter. See Implementing Guaranteed Processing 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 it is not an error for multiple in-process adapters to display the same screen name, users will be unable to differentiate between identically displayed adapter screen names.

The Description and Documentation elements provide summary and detailed information relevant to the adapter.

To invoke the user-written C/C++ code, the Coral8 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. Notice the library name does not contain a trailing ".dll" or ".so". This permits cross-platform loading of libraries that will be handled by the Coral8 engine.

The external name of each of the adapter entry points specifies the name that Coral8 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 the user 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 Coral8 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*> element configures an individual parameter. There may be an arbitrary number of parameters in an ADL file; the exact number depends on the user's needs. The <*Default*>, <*Description*> and <*Required*> elements are mandatory for each parameter. The remaining elements 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* determines the data type of the parameter. See Printing a Parameter for a table that shows which xsi:type corresponds to each internal Coral8 datatype.

The <*Default*> element determines the default value of the parameter. This value gets displayed in Studio on the initial display. When the value is modified, Studio remembers the modified value so it does not have to be reset on subsequent Studio invocations. The <Default> must be of the same data type as the xsi:name, otherwise Studio will issue an error message. Notice that in this example *Port* does not have a default and the <Required> element is true, meaning the user must enter a value.

The <*Description*> provides textual information for the "?" popup help message Studio displays.

The <*Required*> element specifies whether it is absolutely necessary for a user to provide a value for this parameter. If the user provides a <Default> for a <Required> parameter, and does not modify the parameter, then the default will be used. If the user blanks the input field of a <Required> parameter, Studio will highlight the input field to indicate that the user 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 the user for inputting into the field.

The <*MultiLine*> element causes a larger rectangular display to appear on the adapter's Properties View in Coral8 Studio. This would be used for parameters requiring potentially lengthy input strings. An example might be a database query that consumes several lines of input.

Numeric datatypes may use the <*Min*> and/or <*Max*> elements. It is possible to have a <*Min*> without a <*Max*> and vice versa; these elements are independent. The *Port* parameter above must be between 1 and 65536.

Adapter Definition Language (ADL)

Warnings and Tips

1. When you create your own ADL file, make sure that you change only "values", not "tags", inside the ADL file.

2. Do not change the header (the "AdapterDefinition" section) except the direction ("Input" or "Output") and the name of the adapter.
3. 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.
4. When you upgrade your Coral8 Engine, the Coral8 Studio installation program will delete and then re-create the directory that has the ADL files. This means that any ADL files that you have added will be lost! Each ADL file that you create should be backed up in a location outside your `coral8` directory.

Furthermore, 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 Coral8 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

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 Coral8/General/PluginsFolder preference in the server's configuration file. Unless you have changed this preference or installed in a directory other than the default directory, the plugins directory will be:

  ```
  C:\Program Files\Coral8\Server\plugins (on Microsoft Windows)
  ```

  `or`

  ```
  $HOME/coral8/server/plugins (on UNIX-like operating systems)
  ```

- <app path>/plugins

  (i.e. in the plugins subdirectory of the directory in which the application was started)
- <app path>/../plugins

  (i.e. in the plugins directory that is a sibling of the directory in which the application was started)
- ./plugins

  (i.e. in the plugins subdirectory of the current directory)
- ../plugins

  (i.e. 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:

  `Coral8 Repository\<version>\plugins`

  On UNIX-like operating systems, this is:

  `Coral8 Repository/<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\Coral8\Studio\plugins (on Microsoft Windows)
```

or

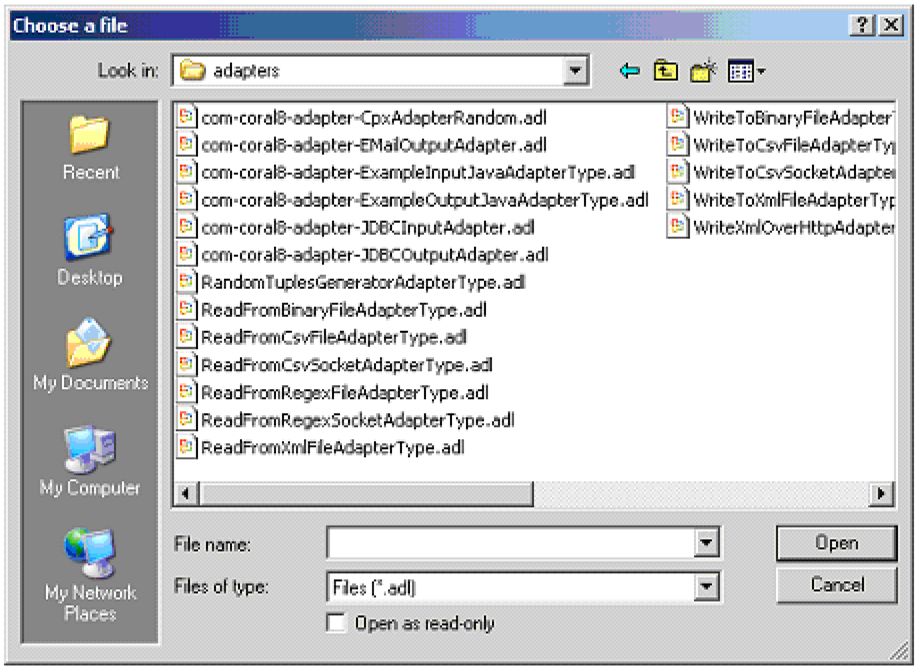
```
$HOME/coral8/server/plugins (on UNIX-like operating systems)
```

If the CCL compiler is unable to find the .adl files after searching the directories specified above, then the compiler issues a message stating that it cannot locate the plugins directory.

Importing New Adapters

In Coral8 Studio, the Attach Adapters Dialog displays a list of input and output adapters that come out of the box with Coral8. When you develop a new in-process adapter, you will typically want it to be included in the list of available adapters that you can attach. There are two main ways to do this. You can copy the .adl file to Studio's "plugins" directory and then re-start Studio. If you don't want to stop and re-start Studio, you may import the ADL file into Studio.

To import a new Adapter Definition, go to the "Tools" Menu and select "Import Adapter Definition..." This brings up the dialog depicted below showing 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.



com-coral8-adapter-CpxAdapterRandom.adl
com-coral8-adapter-EMailOutputAdapter.adl
com-coral8-adapter-ExampleInputJavaAdapterType.adl
com-coral8-adapter-ExampleOutputJavaAdapterType.adl
com-coral8-adapter-JDBCInputAdapter.adl
com-coral8-adapter-JDBCOutputAdapter.adl
RandomTuplesGeneratorAdapterType.adl
ReadFromBinaryFileAdapterType.adl
ReadFromCsvFileAdapterType.adl
ReadFromCsvSocketAdapterType.adl
ReadFromRegexFileAdapterType.adl
ReadFromRegexSocketAdapterType.adl
ReadFromXmlFileAdapterType.adl
WriteToBinaryFileAdapter
WriteToCsvFileAdapterTy
WriteToCsvSocketAdapte
WriteToXmlFileAdapterTy
WriteXmlOverHttpAdapter

File name:
Files of type: Files (*.adl)
Open as read-only

Server Plugins

Coral8 Server's functionality may be extended with one or more "plugins", each of which is a linkable library (.so (shared object) file on UNIX-like operating systems, or .dll (Dynamic Link Library) on Microsoft Windows).

Before you read the rest of this chapter, you should have already read Plugins, which provides an overview of how plugins work.

In the next few sections, we will describe:

- the types of events that may occur;
- the generic plugin that Coral8 provides;
- how to use the generic plugin to implement High Availability (HA) of Coral8 Server Managers.
- authentication-related plugins

Message-Driven Plugins

A plugin is an event handler; in other words, when a particular "event" occurs, specific code within the handler is invoked. That code may, in turn, do almost anything. For example, if the event is that the original manager died and the "backup" has taken over as manager, then the plugin might send 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 Coral8 manager).

Events

The manager writes status information to Server Status streams, which are special Coral8 streams that transmit messages with various server events and metrics. (Each message is sometimes referred to as an "event". You may also think of each message as a row in the stream.) 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.

We currently have two groups of status events: container events that apply to the container server and manager events that apply to the manager server.

Container Events

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 Coral8 Server (Bytes)
ContainerUsedMemory	1/sec	Total memory used by the Coral8 Server (bytes)
ContainerCPUUtilization	1/sec	Percentage of CPU utilized by the Coral8 Server process (1 = 100%)
ContainerCPUTime	1/sec	CPU Time used by the Coral8 Server process since start (Microseconds)
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

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)

The Coral8 Generic Plugin

Coral8 provides a generic plugin that allows you to invoke a specified executable program when a specified status event occurs.

To use this plugin, you update Coral8 Server's configuration file (`coral8-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 Coral8).
- the names of the initialize(), execute() and shutdown() functions.

When you use the generic plugin supplied by Coral8, these preferences are always the same - in other words, the name of the library, and the names of the initialize(), execute() and shutdown functions are always the same.

The next preference, the EventType, is also required. The value that you specify will be one of the valid events listed 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 (a program, a shell script, etc.) that you want to run. The actual value depends upon what action you want to take when the event occurs. IMPORTANT: On UNIX-like operating systems you must specify the full path as well as the program name. On Microsoft Windows, you may specify only the name of the program if the path to the program appears in the %PATH% environment variable.

The parameters that contain command arguments (i.e. command-line arguments to the program specified in the CommandName parameter) are optional and will vary depending upon what action you want to take and what parameters your command requires. You may 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 were 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 will check 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, then processes that have been running longer than the CommandTimeoutSeconds interval will be terminated, so that another process can be spawned in their place. IMPORTANT: CommandTimeoutSeconds has NO EFFECT until MaxRunningProcesses is reached. The default value of CommandTimeoutSeconds is 60. A value of 0 means that processes should not be timed out.

For events (e.g. messages about CPU consumption) that occur once per second, most event-handling programs should finish fairly 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 (i.e. less than the time interval between events, which is 1 second for CPU consumption events).

For events that occur infrequently (such as a container dying), you may not need to worry much about how long these take to run, but you do have to have a good idea of the "worst-case scenario" (i.e. the maximum number of such events that might occur within a specified time) 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 may spawn multiple programs. You'll need to specify a separate event/program pair for each of these - there's no direct way to specify that a single event will run multiple programs. Conversely, more than one type of event may spawn the same program. You'll need to specify a separate event/program pair for each of these - there's no direct way to specify that a single event will run multiple programs. 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.

How to Implement Manager HA with the Coral8 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. Specifically, the new active manager must re-point the generic manager hostname to the 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 Coral8 generic plugin can be configured to respond to this event by calling a program (e.g. nsupdate) that will update the DNS record.

For more information on this topic, please consult the Administrator's 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

Coral8 Engine Enterprise Edition allows users to explicitly permit or deny access to specified resources. For example, you might allow only certain specific users or computers to subscribe to (read from) a stream that contains private financial data.

The Coral8 Administrator's Guide describes the "Actions", "Resources", and "Subjects" that may be restricted. In short, a subject is a particular user, a particular group, or a particular computer. A resource is a particular Coral8 Server object, such as a stream or a workspace. An action may be "read", "write", "create workspace", etc. So, for example, I might have rules that permit user Jane Smith to subscribe to and publish to the stream "StockQuotes", while I allow Patty Jones to subscribe to this stream but not to publish to it, and I might disallow any other user from reading or writing to this stream.

To implement this restriction, you might use a plugin that asks a user to enter her username and password before she is allowed to access a particular resource -- such as subscribe 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, then the server will not allow the user to access that particular resource. You may choose whatever software you wish to use to verify that the password is correct for the username.

Coral8 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 she claims to be.
- htpasswd - this is a simple plugin that allows you to use encrypted password files to determine whether a user is who she 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 (e.g. without recompiling the Coral8 Server).

In addition to using these plugins, you may write your own plugin, which may connect to some other system to verify the password. If you write your own plugin, you will need to specify the library name that contains the plugin, and the names of the initialize(), execute(), and shutdown() functions of the plugin. Please see the documentation of the htpasswd plugin (below) for an example.

For more information on this topic, please see the following:

- User Authentication htpasswd Plugin
- User Authentication LDAP plugin
- User Authentication via Pluggable Authentication Module (PAM)
- the *Coral8 Administrator's Guide* (explains how to configure the Coral8 Server configuration file (`coral8-server.conf`) to tell the server to read the ACL file (`coral8-acl.xml`)
- the *Coral8 Administrator's Guide* (explains some contents of ACL files (Subjects, Resources, and Actions))

For any of these authentication plugins, you must customize the `coral8-server.conf` file in at least 2 places. The first place is shown below; you must de-comment the "ACLFile" preference and, optionally, replace the file name "`coral8-acl.xml`" with another file name if you want to use a different file:

```
<section name="Coral8/Security">
<section name="AccessControl">
<!-- The location of the coral8 access control list file -->
<!-- Default: empty -->
<!-- <preference name="ACLFile"
   value="C:\Program Files\Coral8\Server/conf/coral8-acl.xml"/> -->
```

You must, of course, customize the contents of the coral-acl.xml file (see the Coral8 Administrator's Guide for details).

The second part of the coral8-server.conf file that you must customize will depend upon which plugin you are using. More information is provided in the descriptions of the individual plugins below.

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 `coral8-server.conf` configuration file to specify information about the plugin. Below is an excerpt from the default `coral8-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\Coral8\Server/conf/htpasswd.txt"/>
  <preference name="GroupFilePath"
     value="C:\Program Files\Coral8\Server/conf/htgroup.txt"/>
</section>
```

The initialize and shutdown functions perform just as described in Plugins.

The AuthenticateFunction corresponds to the "execute()" function described in Plugins; this function is called each time that a user gives her username and password in order to access a resource, such as a stream.

This custom entry PasswordFilePath containspairs of usernames and passwords that the plugin uses to determine whether a user is who 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.

Let's look at an example. Suppose that we 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. Here's how to do that:

1. In the ACL (Access Control List) file (named `coral8-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 -- i.e. any name.)
        -->
        <Workspace>.*</Workspace>
      </Resources>
      <Actions>
        <CreateDestroy/>
        <GetStatus/>
      </Actions>
     </Target>
   </Rule>
   ```

   Note that group names, host names, and user names are not case-sensitive.

   Note that values may be regular expressions. In the example above, the "." represents any character, and the "*" is a repeat indicator -- i.e. it 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 `coral8-server.conf` file), enter at least the following:

   ```
   SysAdmins:jsmith
   ```

   You may of course 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 (we specified this file in the PasswordFilePath preference of the `coral8-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 user jsmith tries to create a new workspace, the following will happen:

1. The user starts Studio and attempts to add a new workspace.
2. Studio prompts the user for her username and password.
3. The user enters "jsmith" as the username and "carrot" as the password.
4. The server sees that the server configuration file specifies that the htpasswd plugin should be used, and so the server calls the function specified in the AuthenticateFunction and passes the ID and password that the user typed in.
5. The plugin encrypts the password that the user entered, then looks in the htpasswd.txt file for user "jsmith" and determines that the encrypted password in the file matches the password that the user entered for jsmith. The plugin then returns a value indicating that user has been authenticated as jsmith.
6. The server then looks at the ACL file and sees that although there no rule explicitly permits jsmith to create a workspace, the group "SysAdmins" is permitted to create a workspace.
7. The server then looks in the htgroup.txt file and sees that jsmith is a member of the group "SysAdmins".
8. The server then allows the authenticated user to create a new workspace.

User Authentication LDAP plugin

As with any plugin, you must update the `coral8-server.conf` configuration file to specify information about the plugin. The LDAP plugin supplied by Coral8 follows the standard pattern for LDAP. The parameters that you must specify in the `coral8-server.conf` configuration file are:

- (required) Hostname : e.g. 'ldap.eng.coral8.com'
- (optional) Port : (default 389)
- (optional) Timeout : connections timeout in sec (default: 10)
- (optional) Version : 2 or 3 (default 3)

- (optional) Debug: yes or no (default - no)
- SSL/TLS settings
    - (optional) SslStart : yes or no (default - no)
    - (optional) TlsStart : yes or no (default - no)
    - (optional (not available on Microsoft Windows)) TlsCheckServerCert : yes or no (default - no)
    - (required if TlsCheckServerCert is yes (not available on Microsoft Windows)) TlsCaCert : path to the cert
- (optional) BindDN, BindPassword : if left empty/not specified then use anonymous search
- User settings
    - (optional) UserUidAttr : string (default 'uid')
    - (optional) UserBaseDN : string (e.g. 'ou=People,dc=coral8,dc=com')
    - (optional) UserFilter : string (e.g 'objectclass=posixAccount')
- Group settings
    - (optional) GroupMemberUidAttr : 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.
    - (optional) GroupMemberDnAttr : 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.
    - (optional) GroupBaseDN : string (e.g. 'ou=Groups,dc=coral8,dc=com')
    - (optional) GroupFilter : string (e.g 'objectclass=posixGroup')
    - (optional) GroupLimit: integer - how many groups to return (default : 0 - no limit)

Note that SSL or Transport Layer Security (TLS)-protected connection must be used to prevent user credentials from network analyzers. Please see the Coral8 Administrator's Guide for more information about configuring SSL.

User Authentication via Pluggable Authentication Module (PAM)

Users of Linux and some other unix-like operating systems (including recent versions of Solaris) may use the Pluggable Authentication Module (PAM) system. (At the time PAM was added to Coral8, PAM was not available on MS-Windows systems.)

PAM allows PAM-compatible applications (including the Coral8 Server) to switch authentication methods without recompiling the application. This allows a system administrator to upgrade to a completely different authentication system (such as moving from passwords to retinal scans) 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 may be purchased or may be developed by users.

This document assumes that you already have the PAM modules that you need and therefore we explain only how to configure your Coral8 Server 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 Coral8, 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 Administrator's 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 already read an overview of the PAM system (e.g. chapters 1-4 of the PAM System Administrator's Guide) and understand how PAM works.
- You have already downloaded and installed the PAM files (library files, etc.) that you will need. (Note that 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

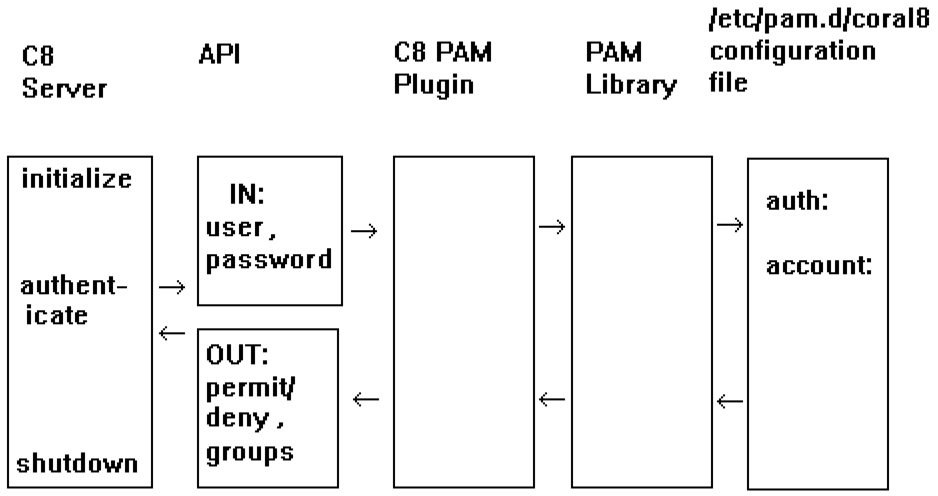
  Please note that access to the `/etc/pam.d` directory or the `/etc/pam.conf` file should be tightly restricted; otherwise, a malicious user could change the configuration files and weaken security.
- You have acquired system administrator privileges on your computer or have other privileges sufficient to allow you to create or modify the `coral8-server.conf` file and the `coral8-acl.xml` file.

  The location of the server configuration file depends upon where you install Coral8 to, but a typical location is:

```
<install-dir>/server/conf/coral8-server.conf
```

A Brief Overview of PAM with Coral8

In the diagram below, you can see the basic components involved when Coral8 uses PAM.



Coral8 Server contains functions designed to work with PAM, including the initialize, authenticate, and shutdown functions. You will need to configure your `coral8-server.conf` file to specify the name of the library that contains these functions. (We will explain this in more detail below.)

The API box indicates what type of information is transmitted from the Coral8 Server to the Coral8 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, and
- 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:

- Read the `/etc/pam.d/coral8` file to determine which configuration method (e.g. password, retina scan, fingerprint, etc.) should be used;
- Call the appropriate functions to authenticate the user (e.g. request password info, request fingerprint info);
- Return a permit/deny value to the C8 PAM Plugin, which, in turn, passes this information back to the Coral8 Server.

The PAM library does more than this, but these are the key high-level tasks as far as the Coral8 Server is concerned.

The `/etc/pam.d/coral8` file specifies which authentication method should be used, and other details. Some of these details are explained in more detail below. For other information, see the internet pages that document PAM.

Configuring Coral8 Server to Use PAM

As with any plugin, you must update the `coral8-server.conf` configuration file to specify information about the plugin. The default `coral8-server.conf` file included with your Coral8 Engine 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 authenticiation.
-->
<!-- <preference name="PasswordPrompt" value="Password: "/> -->
</section>
```

(You should look carefully at your `coral8-server.conf` file to see the exact parameters for your version of Coral8. See also the Coral8 Administrator's Guide.)

By default, this section is commented out in the `coral8-server.conf` file. Make sure that you de-comment it.

As you can see, 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() functions 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 Coral8, the names of the library and the functions in it are known, and you can simply de-comment this section of the `coral8-server.conf` file; you do not need to change the values of the library or function names.

For an explanation of the PasswordPrompt preference, see the section called "PAM Troubleshooting".

Configuring the PAM Service

You create or edit a PAM configuration file to specify which authentication method (e.g. password, fingerprint, etc.) you want to use with Coral8. Coral8 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. We cover key aspects below.

In this document, we assume that you will create

```
/etc/pam.d/coral8
```

Although PAM allows 4 different types of management group rules (account, auth, password, and session), Coral8 uses only two: account and auth.

- auth: this indicates that the user must be authenticated to prove that she is whom she claims to be, and also 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 upon which PAM library module you are using. See 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 all users (this may be useful for testing, but is insecure!).

```
# type    control   module-path    module-arguments
# -----   --------  -------------  ----------------
auth      required  pam_permit.so
account   required  pam_permit.so

```

This configuration file denies all access (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/coral8`, 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 `coral8-server.conf` file's PasswordPrompt preference (which may occur if your password prompt has been localized), then you will need to change the `coral8-server.conf` file's preference to match your actual prompt.

  Explanation: At a certain point in the authentication process, the PAM library calls the Coral8 PAM plugin and passes the system's password prompt. The PAM library expects the Coral8 Server to display the prompt, retrieve the password from the user, and then return the password to the PAM library. However, when Coral8's PAM plugin receives the prompt, Coral8's PAM plugin does not actually display the prompt, but instead returns password information that Coral8 Server already has. If the system's password prompt does not match the PasswordPrompt preference, then Coral8 Server 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.

Unfortunately, if the password prompt sent by the PAM library does not match the password prompt expected by the Coral8 PAM plugin, you will not get a detailed error message. The only symptom you will see will be that authentication fails.

Datatype Mappings

This table shows the approximate mappings between CCL data types, and ANSI data types.

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.

Please note that some database servers (including Coral8) use LONG for a numeric data type, while other database servers use LONG to refer to a character data type.

Recommended Datatype Mappings between CCL and SQL

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 Coral8 Server (e.g. by using the READ FROM DATABASE input adapter, the POLL FROM DATABASE input adapter, or a database subquery) or if you are going to write data to a database server from your Coral8 Server (e.g. by using the WRITE TO DATABASE output adapter or by using the Database Statement ("EXECUTE STATEMENT DATABASE ...").

These recommended mappings are based on our in-house testing. Note that in some cases, other mappings will also work. For example, although we map Coral8 BOOLEAN to MySQL INTEGER, it is likely that MySQL BOOLEAN will also work.

Note also that 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.

We provide 2 tables: The first table shows the mappings between several specific database servers and Coral8. The second table shows the mapping between Coral8 data types and ANSI SQL data types. We recommend that you look in the first table for your database server. If your database server is not listed, then if your database server supports ANSI SQL data types, use the second table.

CCL Type	**DB2**	**MySQL**	**MySQL MaxDB**
BLOB	VARCHAR(N) [7]	VARCHAR(N) [6] [7]	VARCHAR(N) [7]
BOOLEAN	INTEGER	INTEGER	INTEGER
INTEGER	INTEGER	INTEGER	INTEGER
LONG	BIGINT	BIGINT	FIXED(38)
FLOAT	FLOAT	DOUBLE	FLOAT
STRING [3]	VARCHAR(N)	VARCHAR(N)	VARCHAR(N)
INTERVAL	BIGINT	BIGINT	FIXED(38)
TIMESTAMP	TIMESTAMP	DATETIME	TIMESTAMP(6)

XML	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)

Notes:

1. This data type is not part of the ANSI SQL92 specification.
2. Coral8 does not support YEAR-MONTH intervals.
3. On some platforms, Coral8 strings may be limited to 65,535 bytes.
4. Although internally Coral8 supports BLOBs up to 4GB, the practical maximum size is often 2GB due to other limits. For example, Coral8 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. Note specifically that MySQL BLOB is not supported for use with Coral8 BLOB.
7. Note also that when Coral8 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 3 bytes of BLOB data to 4 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.

The table below shows the mappings between Coral8 data types and ANSI SQL data types. If your database server supports ANSI SQL data types, then you should be able to use this table.

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]
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 Coral8 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.

Troubleshooting

This appendix helps you identify and resolve problems that are commonly experienced by users of the Coral8 SDKs.

General Tips

Re-compile in-process adapters, UDFs, and server plugins when you upgrade Coral8 versions. Code that runs inside the server (such as in-process adapters, User-Defined Functions (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 Coral8 engine (e.g. from version 5.0 to version 5.1), you should re-compile that code (and, of course, 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 Coral8ReleaseNotes.txt file in the server "doc" directory. (Note that if you skip over intermediate versions, e.g. 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 Coral8 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.

Visual Studio .NET 2005 updates project-related files (e.g. .vcproj or .sln), and earlier versions of Visual Studio cannot read the updated files. Therefore, upgrading is "irreversible". You would have to re-create the project-related files to downgrade.

Errors When Compiling C-language Adapters, UDFs and RPC Plugins

Problem

You are using MS Visual Studio and you 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)
```

If you see this message, 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 (go to the menu and click on "View", then click on "Solution Explorer"); the Solution Explorer window will open up on the right-hand side of the MS Visual C IDE window. Click on the .cpp file (e.g. 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.

Error Messages When Compiling Java-Language Adapters

Problem

When you compile, you see one or more error messages similar to the following:

```
MapPublisher1.java:1: package com.coral8.toolbox does not exist
import com.coral8.toolbox.Toolbox;
MapPublisher1.java:33: cannot find symbol
symbol : variable Toolbox
location: class MapPublisher1
Toolbox.publishRow(cclUrl, 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\Coral8\Server\sdk\java\c8-adapters.jar
C:\Program Files\Coral8\Server\sdk\java\c8-sdk-java.jar
```

Errors When Starting the Server

Problem

When Coral8 Server is attempting to start, it exits without apparent reason.

There may be another server running that is using the same port number.

Problem

When the Coral8 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) 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, please make sure that you compile and link in-process adapters and User-Defined Functions with a Microsoft Windows development system such as Microsoft Visual Studio, rather than with the C compiler supplied with cygwin. Note that 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

You will usually see these messages displayed during execution, typically displayed by Coral8 Studio, for example.

Problem

Coral8 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 Coral8 Server cannot find the entry point in the adapter library.

Notice this error also is printed in the Coral8 Server log. Please check the following to see if any of them might be causing your problem.

- Has the library been copied to the bin subdirectory of the Coral8 Server directory?
- Is the entry point name spelled correctly? Remember also that these names are case sensitive.
- If the user is debugging with Visual Studio, the debug and release versions can be in different directories.
- Has the user copied adapter files to the correct plugins directories?
- Has the user copied 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/Coral8 Repository/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" (e.g. 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. Make sure that you copy it to the "bin" directory of the server, e.g.

  `C:\Program Files\Coral8\Server\bin`

  or

  `/home/<userID>/coral8/server/bin`

- Your .udf file might not specify the same library name and function name as the .dll/.so file use.
- The "case" (e.g. 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/Coral8
Repository/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/Coral8/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 (e.g. MSVCR8D.DLL) rather than the non-debug version (e.g. 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 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.
- You may need to turn off debugging. In the Microsoft Visual C compiler, you can turn off debugging by doing the following 2 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, i.e. . "Multi-threaded DLL (/MDd)"
    2. Check the pre-processor 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/Coral8
Repository/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/Coral8/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 (e.g. MSVCR8D.DLL) rather than the non-debug version (e.g. 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.
- You may need to turn off debugging. In the Microsoft Visual C compiler, you can turn off debugging by doing the following 2 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, i.e. . "Multi-threaded DLL (/MDd)"
  2. Check the pre-processor 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.

Don't forget: 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\Coral8\Server\sdk\java\lib
```

See Setting Up Your Environment, 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, of course.)

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 Coral8 Studio, you can see the URL by clicking on the stream and then clicking on the "Properties" tab for that stream.

Problem

If C8OpenStreamConnForReading() or C8OpenStreamConnForWriting() returns a value other than C8_OK, check the following possibilities:

1. Is the URI spelled correctly? (Remember that the URI is case-sensitive.)
2. Is the Coral8 Server running? Is it accessible from the test client machine.
3. Has the Coral8 Studio initialized the stream? If Coral8 Studio is not running, or if the query module has not been loaded into 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, are permissions set correctly to allow reading and writing?

Problem

If you try to use a TitleRow (i.e. a list of the column names in the stream schema) and parts of the title row are processed as though they were data, then remove the TitleRow from the input.

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 server/bin directory or whatever directory the library file is in.

To solve the problem, update your LD_LIBRARY_PATH and then restart the server.

On both MS-Windows and Unix-like operating systems, 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 (User-Defined Function):

```
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.

You may not have copied the .udf file to Studio's plugins directory. (You might have copied the file only to the server's plugins directory.)

Check that you used the correct values

HTTP and SOAP Plugin Configuration

The `coral8-services.xml` configuration file contains preferences related to interfacing Coral8 with other systems, including:

- Database Servers
    - MySQL, PostgresSQL, SQLServer via ODBC
- RPC (Remote Procedure Call) servers

Coral8 Server treats an external data source (or data destination) as a "service". For each service, the `coral8-services.xml` file must contain a unique "service name", along with the information required to access that service (e.g. username, password, etc.). The *Coral8 Administrator's Guide* describes how to configure your `coral8-services.xml` file with the service name and most other information needed to access remote database servers and RPC servers.

The Administrator's Guide does not describe configuration preferences for the HTTP plugin and the SOAP plugin, either of which may 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 `coral8-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". Defaults to 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 `coral8-services.xml` file:

All the HTTP plugin preferences	Same as for the HTTP plugin
SoapUri	Required: indicates the URI of the SOAP resource, for example, `urn:StockPriceService`
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" (the default) or "1.2".
SoapMethodNamespace	Optonal: 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	Optonal: indicates the namespace to use for the values node. Defaults to the same value as **SoapMethodNamespace**. If you set this to the empty string, then the values node is written without a namespace.
SoapEncodingStyle	Optional: "Document" (the default) or "Rpc", specifying the encoding style for the SOAP request and response.
SoapXsdNamespace	Optional: specifies the "xsd" namespace URI. The default value is `http://www.w3.org/2001/XMLSchema` However, old SOAP servers may require the old version: `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 `http://www.w3.org/1999/XMLSchema-instance`
SoapDotNet20Compat	Optional: TRUE or FALSE, indicating whether .NET 2.0 compatibility mode should be enabled or disabled. .NET 2.0 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. (The SOAP format requires the "T" before the hour and the "Z" before the time zone information. These extra characters are compatible with the Coral8 format; Coral8 simply treats them as literals.)

The remote service entry may also be configured for a number of additional settings, as described in the Coral8 Administrator's Guide section titled "Setting Optional Preferences for Services".

To learn the syntax of RPC and Database statements and subqueries, please see the Coral8 Reference Guide.

Stream URIs

You should read this section if you are going to use any of the following features:

- Out-of-process adapters or any other program that uses the Coral8 SDKs
- Distributed queries (including pipelined queries and parallel queries)
- High Availability (HA)

You should also read this section if you need to make URIs "portable" across servers.

What Is a URI and What Is It Used For?

Each Coral8 stream has a unique stream URI ("Uniform Resource Identifier"), which provides enough information that a program (such as an out-of-process adapter) can connect to that stream. (If you are already familiar with URLs, it may be helpful to know that a URI is a lot like a URL.) You must also specify a URI when you connect streams in different projects.

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 will read. Any other program that uses a Coral8 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 a stream. (This is explained in more detail in the section titled URIs and Distributed Queries.)

Another use of stream URIs is with the High Availability (HA) feature. When using HA, Coral8 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 thus a program "talking to" or "listening to" a stream may need to re-connect to that stream since the stream may now be running on a different computer. Thus, Coral8 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: Coral8 URI naming follows the RFC 3986 standard for URIs. A CCL URI is a URI with the "ccl" scheme.

Logical vs. Physical Stream URIs

Coral8 stream URIs may be "logical" or "physical" URIs. User-written programs, such as out-of-process adapters, generally specify a logical URI, which the manager resolves (translates) into a physical URI. The physical URI includes information about which container the stream is running on. (If you are not already familiar with managers and containers, see the explanation in the High Availability (HA) documentation.)

- Logical URI -- Logical URIs start with "ccl://".
- Physical URI -- 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", i.e. URIs that are relative to the local Coral8 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 Coral8 Engine, a stream will be split into separate "instances" and the addresses of the stream instances will be the same except for a tilde ("~") 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
```

(You should start with the number 1, and increase by 1 for each additional stream you will 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. You will wind up with 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.

Note that 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 (i.e. sending data from an output stream in one project to an input stream in another project), you will 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 Coral8 Studio, see the Coral8 Studio Guide.

URIs and High Availability

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.

How to Find the URI of a Stream

The easiest way to find the URI for a particular stream is to use Coral8 Studio and follow the steps below to find the stream's URI in Coral8 Studio's Properties View for the stream.

1. Start Studio (if you have not already started it).
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 Coral8 Studio).
4. Click on the stream to display the "Properties" View for the adapter.
5. Read the "Stream URI" and the "Http URI" from the Properties View.

Summary

Out-of-process adapters -- Out-of-process adapters (and other programs that access Coral8 Server through a Coral8 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, etc.

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 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 that new physical URI.

If a manager is lost because its entire host computer shut down, and if a computer with a passive (i.e. "backup") 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 will communicate with the DDNS process and will "take" the hostname of the original computer. This means that if there are other (non-Coral8) processes that were running on the old (dead) host, and if those processes were accessed from other computers, anyone trying to contact those processes will be confused when the replacement host comes up under the old hostname and those other processes are missing.

Connecting to Streams over a Network

As we mentioned earlier, the Coral8 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 (Comma-Separated Values)

This chapter describes how to connect directly to a Coral8 stream over the network and how to publish or subscribe to a stream after you've connected to it directly.

This chapter contains these sections:

- Data Stream URI
- Subscribing To Data Stream
- Publishing To Data Stream
- Binary Data Stream Format
- CSV Data Stream Format
- XML Data Stream Format

Data Stream URI

Each native data stream has a URI (Uniform Resource Identifier) 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 Coral8 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 Coral8 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 Coral8 data stream, the subscriber must perform an HTTP GET request to the Coral8 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.)

NOTE: The "\r\n" represents the 2-character sequence carriage return and line feed, not the 4-character literal "\r\n".

NOTE: The header should be followed by a blank line terminated with a carriage return and line feed.

The client should expect to receive an HTTP response which 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 would look 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 (should be ignored by subscriber).

As before, note that the "\r\n" represents the 2-character sequence of a Carriage Return followed by a Line Feed.

Note that the Coral8 HTTP response is different from most HTTP responses from entities other than Coral8 Server. The output data stream is a series of messages that may continue indefinitely. Thus 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 (not a single message) and each of those messages should be dealt with when the message (i.e. the complete output row) arrives. The recipient does not wait until the "end" of the HTTP response to start processing the messages in it.

We will now look at an example of an HTTP request and response.

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).

And the subscriber would receive 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 Coral8 data stream, the publisher must perform an HTTP POST request to the Coral8 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 formattiing 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).

Data Stream Formats

This section describes each of the three Coral8 data formats:

- binary
- CSV (Comma-Separated Value)
- XML

Binary Data Stream Format

Binary Data Stream Format is the primary data stream format for Coral8 Engine. It is very fast and efficient. The format description will be available in future versions of this document.

CSV Data Stream Format

Coral8 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

[foo] bar

the "foo" is optional, while the "bar" is required. If the "foo" 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. E.g. "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 may 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.  E.g.
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, Coral8 reads and writes the NEWLINE using the convention for the current platform (e.g. 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) which specifies whether the title row is present or not;
- TimestampColumn (true or false) which specifies whether the first column is the message's timestamp column or not;
- TimestampColumnFormat (string) which 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 sole 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. (Note also that if you specified that the row timestamp should be included in the output, then the row timestamp will always be the first column.)
- Not surprisingly, if you specify that the output should include a title row, the names of the columns in that title row will be exactly the same as the names in the schema.

XML Data Stream Format

Coral8 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. Note that there is no single root element in the stream. Each Tuple element must conform to the following schema:

```
<xs:schema xmlns="http://www.coral8.com/cpx/2004/03/"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.coral8.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 that 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.coral8.com/cpx/2004/03/">
xmlns="http://www.coral8.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.coral8.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.coral8.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.coral8.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.

Coral8 Adapters

This chapter describes the input adapters and output adapters that Coral8 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 Coral8 Adapters

Most adapters supplied by Coral8 have user-settable properties (also called "parameters "). Some of these properties have default values and some do not. You should 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 Coral8 Studio. See the *Coral8 CCL Reference* and the *Coral8 Studio Guide* 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 `coral8-server.conf`.

Setting the Base Folder for File Input/Output Adapters

You specify a directory as the *adapters base folder* when you install Coral8 Server. You can change the base folder by editing the Coral8 Server configuration file and modifying the value for the "Coral8/Adapters/ReadWriteBaseFolder" preference. The adapters base folder tells Coral 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 Coral8 Server (this prevents Coral8 Server from accidentally interfering with files used by other software).

Adapters supplied by Coral8 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 Coral8 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**. Coral8 Server replaces **$BaseFolder** with the value of the "Coral8/Adapters/ReadWriteBaseFolder" preference in the server configuration file. It replaces **$ProjectFolder** by concatenating **$BaseFolder** with the path to your project **.cpp** file on the client, *relative to the Coral8 repository*. The repository is the directory Cora8 Studio uses for its preferences file, environment file, security file, the examples directory, and its temporary directory. If you use the Coral8 Eclipse Plug-In, the workspace directory is equivalent to the repository directory. Note that, by default, if you install Coral8 Server and Coral8 Studio on the same computer, the repository and the adapters base folder are set to the same directory. If you create your project in a directory outside of the repository, then Coral8 Server replaces **$ProjectFolder** with the absolute path to the project directory without inserting **$BaseFolder**.

Example

For example, say that the repository on your client Windows computer is set to

> **C:\Documents and Settings\jsmith\My Documents\Coral8 Repository\5.5**

You create a project under the repository in the directory **MyProjects\Project1**. The full path to your project on the client computer is then

> **C:\Documents and Settings\jsmith\My Documents\Coral8 Repository\5.5\MyProjects\Project1**

You 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 Coral8 Server, say that you've set your adapters base folder to **/var/Coral8/BaseFolder**.

When you compile and run your project on Coral8 Server, it will look for the data file with the following path:

> **/var/Coral8/BaseFolder/MyProjects/Project1/Data/MyData.csv**

This is the adapters base folder on the server (**/var/Coral8/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 Coral8 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, Cora8 Server would look for the data file with this path:

> **/var/Coral8/BaseFolder/Data/MyData.csv**

This is just the adapters base folder followed by the rest of the path and file name specified in the project.

Reading and Writing BLOB Data

This section provides information about reading and writing BLOB data via adapters.

The in-process adapters supplied by Coral8 handle the BLOB data type. When data is read or written in Comma Separated Values format (e.g. 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). Since CSV files themselves are limited to 2GB (Coral8 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 (1GB - 1).

Setting Up the Environment for Java Adapters

Adapters that are written in the Java Programming language, including

- JDBC Input Adapter
- JDBC Output Adapter
- JMS Input Adapter
- JMS Output Adapter
- JMS Reliable Adapter (Guaranteed Delivery)
- Java Email Output Adapter

must have the environment (including the CLASSPATH environment variable) set properly. For more information, see Setting Up Your Environment. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.

Setting Up the Environment for the JMS Adapters

This section describes how to configure your system to use the Coral8 JMS (Java Messaging Service) Adapters. Coral8 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 (Java Messaging Service) adapters, you will have to install and configure some software, including a JMS server. The Coral8 JMS adapters are 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 may use a different JMS server.

Note: This section of the manual focusses on using a Coral8 JMS Output adapter, but most of the steps are the same for a Coral8 JMS Input adapter.

Prerequisites

You will need the following:

- Coral8 Server and 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 a New JMS Server and Deploy It.

The instructions for this depend upon which JMS server you are using, and are not included in this Coral8 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 Coral8 JMS Adapter (which is a JMS client) can connect to the JMS server.

Create a Connection Factory appropriate for your JMS server. (The exact instructions for doing this may depend upon 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 will give the Coral8 JMS adapters so that they can look up this JMS Connection Factory.

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.

Note: Some JMS Servers can have destination queues and/or topics. Coral8 publish/subscribes only to topics, not queues.

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 Coral8 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.

Configuring and Setting Up Coral8

This section describes how to configure and set up Coral8 to work with the JMS adapters.

Coral8 Studio

Using Coral8 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 may use whatever data source you wish.) Using the JMS Output Adapter, we will subscribe to this stream and publish the stream data to the desired JMS Topic destination.

Coral8 JAR files

You will need some Coral8 .jar files. These .jar files are in installed on your system automatically when you install Coral8 Server.

Testing the Coral8 JMS Adapter

1. Make sure that you have all of the required .jar files:
    - c8-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 will 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.

NOTE: Although most of the required .jar files already reside in C:\Program Files\Coral8\Server\sdk\java\lib, a few are not found there. The exceptions are listed below:

- The location of the .jar file for your JMS server depends upon which web server you are using. For example, weblogic.jar is found in your BEA\weblogic81\server\lib directory
- c8-adapters.jar and c8-sdk-java.jar are found in the Coral8\Server\sdk\java directory

2. 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 Coral8 Server is started. To start the server, go to Start -> All Programs->Coral8->Server->Coral8 Server
   C. Make sure that your CCL Module is running. Do this by starting the Coral8 Studio and then running your CCL module from within Coral8.
3. Start the JMSOutputAdapter, so that messages from Coral8 get sent to the desired JMS Server topic destination.
   A. Set your CLASSPATH so that it includes all the jarfiles listed in Step 1 of this section.
   B. Give the command to start the JMSOutputAdapter. 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. Naturally, 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/coral8/adapter/JMSOutputAdapter
--messageType=TextMessage
--topic=weblogic.jws.jms.MyJMSTopic
--factoryName=weblogic.jws.jms.TopicConnectionFactory
--factoryClass=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 **--factoryClass** (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 Coral8 stream you want to subscribe to. Note: 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 Coral8 Server. The host and port for the Coral8 Server are specified as part of the c8.baseHostPort.

4. If the JMSOutputAdapter started successfully, you should see the following output:

```
Connecting to CPX server...
Opening a subscription...
Listening for messages...
```

5. Check to make sure that the messages arrive at the topic destination. The way to do this depends upon which JMS Server you are using.

Shutdown Sequence

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 Coral8 module is shut down before the output adapter is stopped, the adapter will display error messages and it will seem like there is some problem with the adapter when the adapter merely cannot find the stream to subscribe to.

Reading, Writing, and Converting Timestamps

Internally, timestamps are always stored as the number of microseconds since midnight January 1, 1970 UTC/GMT. (We refer to this time 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 (e.g. "2007-12-31 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 (e.g. "YYYY-MM-DD HH24:MI:SS.FF **TZD**") then timestamp values should be supplied with a time zone (e.g. "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 (e.g. "YYYY-MM-DD HH24:MI:SS.FF") is supplied, but the format specifier 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 then you 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", which is of course 8 hours before midnight January 1, 1970.

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 (e.g. 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 probably 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 how to specify timestamp formats, see the appropriate appendix in the Coral8 Reference Guide.

For more information about time zones, see *Daylight Savings Time and the Coral8 Time Zone Database* in this manual.

Adapters Supplied by Coral8

This section describes the built-in input adapters and output adapters provided by Coral8.

We start with 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, e.g.

```
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" or "(Not Applicable)" as their adapter type.

Adapter Data Type	**Read / Write**	**Adapter Type (used in ATTACH ADAPTER statement)**
Atom	Atom Reader N / A	AtomReaderAdapterType N / A
Binary file	Read from binary file Write to binary file	ReadFromBinaryFileAdapterType WriteToBinaryFileAdapterType
Comma-Separated Values (CSV) File	Read from CSV (Comma-Separated Value) File Write to CSV (Comma-Separated Value) File	ReadFromCsvFileAdapterType WriteToCsvFileAdapterType
Comma-Separated Values (CSV)	Read from CSV (Comma-Separated Value) Socket	ReadFromCsvSocketAdapterType WriteToCsvSocketAdapterType

Socket	Write to CSV (Comma-Separated Value) Socket	
Database: Poll From Database (see also JDBC)	Poll From Database Write to database	ReadFromDBAdapterType WriteToDBAdapterType
Database: Read From Database (see also Database, JDBC)	Read From Database N / A	ReadFromDatabaseAdapterType N / A
Email: Send Email Out (SMTP)	N / A Send Email Out (SMTP)	N / A OpenSourceEmailAdapterType
Email: Java Send Mail	N / A Java Send Mail	N / A N / A
Ganglia	Ganglia Reader N / A	GangliaReaderAdapterType N / A
Java Messaging Service (JMS)	JMS Input Adapter JMS Output Adapter	N / A N / A
JDBC (see also Database)	JDBC Input JDBC Output	N / A N / A
Random Tuples Generator	Random Tuples Generator N / A	RandomTuplesGeneratorAdapterType N / A
Regular Expression File	Read from Regular Expression File	ReadFromRegexFileAdapterType

	N / A	N / A
Regular Expression Socket	Read from Regular Expression Socket N / A	ReadFromRegexSocketAdapterType N / A
RSS	RSS reader N / A	RssReaderAdapterType N / A
SNMP	SNMP Get SNMP Set	SNMPGetAdapterType SNMPGetAdapterType
SNMP V1 Trap	N / A SNMP Send V1 Trap	N / A SNMPV1SendTrapAdapterType
SNMP V2 Trap	N / A SNMP Send V2c Trap	N / A SNMPV2SendTrapAdapterType
Sybase RAP	N / A Sybase RAP	N / A N / A
XML File	Read from XML File Write to XML File	ReadFromXmlFileAdapterType WriteToXmlFileAdapterType
XML Socket	Read from XML Socket Write to XML Socket	ReadFromXmlSocketAdapterType WriteToXmlSocketAdapterType
XML over HTTP	N / A Write XML Over HTTP	N / A WriteXmlOverHttpAdapterType

Adapter	**Adapter Type (used in ATTACH ADAPTER**

	statement)
Atom Reader	AtomReaderAdapterType
Binary: Read from binary file	ReadFromBinaryFileAdapterType
Comma-Separated Values (CSV): Read from CSV File	ReadFromCsvFileAdapterType
Comma-Separated Values (CSV): Read from CSV Socket	ReadFromCsvSocketAdapterType
Database: Poll From Database adapter	ReadFromDBAdapterType
Database: Read From Database adapter	ReadFromDatabaseAdapterType
Ganglia Reader	GangliaReaderAdapterType
JDBC Input	N / A
JMS Input Adapter	N / A
Random Tuples Generator	RandomTuplesGeneratorAdapterType
Regular Expressions: Read from Regular Expression File	ReadFromRegexFileAdapterType
Regular Expressions: Read from Regular Expression Socket	ReadFromRegexSocketAdapterType
RSS reader	RssReaderAdapterType
SNMP Get	SNMPGetAdapterType
XML: Read from XML file	ReadFromXmlFileAdapterType

Adapter	**Adapter Type (used in ATTACH ADAPTER statement)**
Binary: Write to binary file	WriteToBinaryFileAdapterType
Comma-Separated Values (CSV): Write to CSV File	WriteToCsvFileAdapterType
Comma-Separated Values (CSV): Write to CSV Socket	WriteToCsvSocketAdapterType
Database: Write to database	WriteToDBAdapterType
Email: Java Send Mail	N / A
Email: Send Email Out (SMTP)	OpenSourceEmailAdapterType
JDBC Output	N / A

JMS Output Adapter	N / A
SNMP Set	SNMPGetAdapterType
SNMP Send V1 Trap	SNMPV1SendTrapAdapterType
SNMP Send V2c Trap	SNMPV2SendTrapAdapterType
XML: Write to XML file	WriteToXmlFileAdapterType
XML: Write XML Over HTTP	WriteXmlOverHttpAdapterType

Adapter Properties (Parameters). 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, e.g. the "Filename" in the ATTACH ADAPTER statement below.

  ATTACH INPUT ADAPTER someName ... PROPERTIES Filename = 'xyz.csv' ...;
- Type - the data type, e.g. STRING, INTEGER, etc.
- Description

In many cases, the property's screen name and "AA" name are the same.

Atom Feed Reader Input Adapter

The ATOM feed reader is designed to allow you to get information from ATOM data sources. ATOM data sources allow connections via URL and send their information in a special XML format.

The retrieved information will be inserted into a Coral8 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 will be 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.
Refresh interval	RefreshInterval	Interval	How often to query the specified URL for data. In microseconds, unless qualified with Interval formatting (see "Time Literals" in the *Coral8 CCL Reference* for more information). Optional. Defaults to 1 minute.
Timestamp Format	TimestampFormat	String	The format of the timestamp. See "Timestamp Format Codes" in the *Coral8 CCL Reference* for more information. Optional. 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 Coral8 Server installation directory.

E.g. on Microsoft Windows, the directory is typically:

```
C:\Program Files\Coral8\Server\examples\Web\LiveJournalAlerts
```

On UNIX-like operating systems, the directory is typically

```
/home/<username>/coral8/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 Coral8 binary file. This adapter is used for internal testing only.

DO NOT USE THIS ADAPTER.

Binary: Write To Binary File Adapter.

A *Write To Binary File adapter* writes received Rows to a Coral8 binary output file. This adapter is used for internal testing only. **DO NOT USE THIS ADAPTER.**

Comma-Separated Values (CSV): Read From CSV File Adapter.

A *Read From CSV File* adapter reads Rows from a CSV (Comma-Separated Values) 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. 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 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. 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 greater than 1, then 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, then repeat forever. If the loop count is left blank, the default value (1) will be used.
Rate	Rate	Float Min: 0.000001	If this property is non-zero, then the adapter will read data from the input file at the specified rate (per second). Any row timestamps in the input file will 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	UseCurrentTimestamp	Boolean Default:	If true, the adapter

timestamp to current time		false	overrides the row timestamp specified in the file with the current system time.
File has timestamp column	TimestampColumn	Boolean Default: true.	Every row must have a row timestamp that indicates the time that the row was created. The row timestamp may be specified by the user in the input data (e.g. in the CSV file), or this timestamp may be set by the adapter at the time that the row is sent to the server. If "File has timestamp column" is true, then 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 Programmer's Guide for more information about row timestamps.)
Timestamp column format	TimestampColumnFormat	String	If "File has timestamp column" is set to true, then 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. For more information, see Reading, Writing, and Converting Timestamps. Note that 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.
Timestamp Base	TimestampBase	Timestamp	If this property is set, then when the adapter starts, it behaves as though it had started at the time specified by the Timestamp base. This affects the time that the first row in the input file has 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 (i.e. 5 seconds after the timestamp base), then 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 you synchronize data in multiple streams (i.e. multiple input files). 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 then 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 should be the same as the order of the Row descriptor fields.
Field Separator Character	CsvDelimiterString	String Default: , (comma)	This indicates which character(s) separate the fields in the file. In most cases, you will leave this at the default value, which is the comma. However, you may

			specify a different separator character(s), such as a vertical pipe symbol ("|") etc. Note that if you specify multiple characters, then 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 (tuple) of data must be split across multiple input lines, then you may 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-whitespace character preceding a newline. See the discussion below for an example. Note that if you specify multiple characters, then EACH of those characters is treated as a line continuation character; the adapter does not look for a multi-character line-continuation indicator. Note that 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.
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. Note that if you specify multiple characters, then 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, then 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) 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. Note that 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 if a string field is "empty" (i.e. if there is nothing between the field separator characters for that field), then the value will be 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, e.g. '$', '|', 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`
Escape Characters	CsvEscapeCharacters	String Default: \	The character or characters used to escape the special meaning of other characters, such as column separators and

			quotes.

Note that each input stream has a property (see the stream's Properties tab in Coral8 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 Coral8 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 Coral8 Server running on the same computer as Coral8 Studio, or as specified in the preferences for Coral8 Studio. For more information, see Setting The Base Folder For File Input/Output Adapters. 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 Coral8 Server. The Base Folder setting for the Server is specified during the installation process, and may also be changed later in the Server's `coral8-server.conf` file. The base folder setting for Coral8 Studio may be set from the Tools->Settings command on the Studio menu.

The "Edit" button for the filename property. The edit button opens an editor that will allow you to edit the 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", then Studio will open 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 will be the one specified by the operating system's file associations, and on UNIX-like operating systems Coral8 Studio will open the editor specified by the EDITOR environment variable.

When you click the Edit button, Coral8 Studio will look for the file in the Coral8 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.

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 allows 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 will 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.

More Detailed Descriptions of 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. The minimum value for Rate (if you specify it) is 0. The maximum rate you can enter is approximately 2 billion (rows per second), but of course the practical maximum depends upon the speed of your computer.

Furthermore, the exact time that rows are sent will depend upon low-level factors such as task switching, so the actual rate at which rows are sent will be only approximately the same as the rate that you specify.

Note that 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 will overwrite the row timestamp (if any) in the input file.

Note that setting the rate does not change the row timestamps in the rows.

File has timestamp column, and *Timestamp format 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 will be sent based on the differences between row timestamps. E.g. if the second row's row timestamp is 5 seconds after the first row's row timestamp, then the second row will be sent 5 seconds after the first row (if no other property, such as Rate, overrides this). If you are using the Accelerated Playback feature, then the actual time that the second row is sent will depend upon the Accelerated Playback rate.

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, then 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.

The "Timestamp format column" property allows you to specify the format of the row timestamp column (e.g. '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 manual for a description of the valid formats for TIMESTAMP data type. See *Daylight Savings Time and the Coral8 Time Zone Database* and Reading, Writing, and Converting Timestamps for more information about time zones.)

The property Timestamp Format Column 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, then the server will assume that the row timestamp is specified in microseconds since midnight January 1, 1970 GMT.

Timestamp Base

The timestamp base allows you to minimize a common problem, which is that 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. Since 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", 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, i.e. 2006-02-01 09:00:05. When you start processing the data, you'd like 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. E.g. 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 will be interpreted according to the setting in the Timestamp Column Format (e.g. "YYYY-MM-DD HH24:MI:SS.FF") if that field is set.

Note that Timestamp base is ignored if "Rate" is set.

Set Timestamp to Current Time

If "Set timestamp to current time" is set to true, then Coral8 sets each row's row 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", then Coral8 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.

Note that 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 (relative timing) 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 have timestamps 5 seconds apart. If you'd like 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, make sure that you 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 will send 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 (e.g. if the row timestamps are 5 seconds apart then the rows will be sent 5 seconds apart).

Loop Count

Note that 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.)

Note

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.

Line Continuation Character

Normally, each row is on a single line, i.e. each row is terminated by a newline. (The ReadFromCSVFile adapter provided by Coral8 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 (row), 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

You may use almost any ASCII character(s) that you want for each of these purposes; however, please 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

The format of INTERVAL values inside a CSV or XML 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]]] (e.g. 1 02:03:04.000005)
- [D day[s]][ ][HH hour[s]][ ][MI minute[s]][ ][SS[.FF] second[s]] (e.g. 1 day 2 hours 3 minutes 4.000005 seconds)

For a complete list of valid formats for INTERVAL values, search for 'INTERVAL Literals" in the CCL Reference.

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 (CSV): Write To CSV File Adapter.

A *Write To CSV File adapter* writes received rows to a CSV (Comma-Separated Values) 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 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. 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 then 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.
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.

Write title row	TitleRow	Boolean	If this flag is set to true, then 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, 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 from 00:00:00 Jan 1, 1970 UTC/GMT. For more information, see Reading, Writing, and Converting Timestamps.
Flush each row	FlushAfterEachRow	Boolean	If set to true, the adapter writes ("flushes") each row to disk as soon as possible. If set to false, the adapter may wait until multiple rows are ready to be

			written to disk and then write them as a group. Not surprisingly, flushing after each row may reduce overall performance (throughput).
CSV delimiter string	CsvDelimeterString	String	The character that separates columns in the output. Defaults to a comma (,).

The "Browse" button for the filename property. The Adapter Properties screen in Coral8 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 Coral8 Server running on the same computer as Coral8 Studio, or as specified in the preferences for Coral8 Studio. For more information, see Setting The Base Folder For File Input/Output Adapters. 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 Coral8 Server. The Base Folder setting for the Server is specified during the installation process, and may also be changed later in the Server's `coral8-server.conf` file. The base folder setting for Coral8 Studio may be set from the Tools->Settings command on the Studio menu.

The "Edit" button for the filename property. The edit button brings up an editor that will allow you to view the file whose name you entered into the filename field. If the file's extension is "csv" or "xml", then Studio will bring 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 will be the one specified by the operating system's file associations, and on UNIX-like operating systems Studio will bring up the editor specified by the EDITOR environment variable.

When you click the Edit button, Coral8 Studio will look for the file in the Coral8 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 open the file.

Format of TIMESTAMP Values In the CSV 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).

Format of INTERVAL Values In the CSV File. When Coral8 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 attempts to open a TCP connection to a given address (specified through Host and Port properties) and, once connection is established, 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	Whether to expect a title row (default: true)
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 the timestamp may be specified by the user (e.g. if running a

			simulation). If the "Data has timestamp column" flag is true, then 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 the adapter will send 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, e.g., 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. In most cases, you will leave this at the default value, which is the comma. However, you may specify a different separator character(s), such as a vertical pipe symbol ("|") etc. Note that if you specify multiple characters, then 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 (tuple) of data must be split across multiple input lines, then you may 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-whitespace character preceding a newline. Note that if you specify multiple characters, then EACH of those characters is treated as a line continuation character; the adapter does not look for a multi-character line-continuation indicator. Note that if the line continuation field is empty, no line continuation will be possible. This allows data with wildly varying input characters to be properly processed.
String Quote Characters	CsvQuoteCharacters	String Default: ' (single quote) and " (double	By default, both the single quote and the double quote characters may be used to

		quote)	delimit a string. Note that if you specify multiple characters, then 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, then 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)

			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. Note that 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 if a string field is "empty" (i.e. if there is nothing between the field separator characters for that field), then the value will be 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, e.g. '$', '|', 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`
Escape Characters	CsvEscapeCharacters	String Default: \	The character or characters used to escape the special meaning of other characters, such as column separators and quotes.

Note that 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 attempts to open a TCP connection to a given address (specified through Host and Port properties) and, once connection is established, 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	CsvDelimeterString	String	The character that

delimiter string			separates columns in the output. Defaults to a comma (,).

Database: Poll From DB Input Adapter

The *Poll From DB* (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.

The user may choose to read the entire table each time, or read only the rows that have been added since the last time that the user 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 Coral8 Studio, and properties for this adapter are set using Coral8 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	The Query is the SQL query (e.g. SELECT statement) that you want to run on the external database server. This field is required. Note that the order of fields that you retrieve should match the order of the fields in the stream to which you are writing. E.g. 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	This indicates which database to execute the

			query on. This is actually be the name of a "service" that is defined in the `coral8-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. This field is required.
Poll Interval	PollInterval	Interval	Poll interval. This indicates how frequently Coral8 Server should check the external database server for changes. The value may be a number of microseconds or may use standard INTERVAL values such as "100 milliseconds" or "1 hour" (without quotes, of course). This field is required.
Counter field	CounterField	Integer or Long	This is the name of the counter/sequence number field. (Note that this is the name of the field in the Coral8 schema, not the name of the field in the external table. This is explained in more detail later in this section.) This column's data type should be compatible with Coral8'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 will retrieve only records whose "counter" field contains values greater than the value specified in this Initial Value field. (Data values equal to the Initial Value will NOT be retrieved.) Note that the initial value of the Counter Field may be either a LONG or an INTEGER, depending upon the data type of the Counter Field.
Timestamp Field	TimestampField	String	This is the name of the timestamp/datetime column. (Note that this is the name of the field in the Coral8 schema, not the name of the field in the external table. This is explained in more detail later in this section.) Note that this column's data type should be compatible with Coral8's TIMESTAMP data type. (See *Datatype Mappings* for tables that show which database data types correspond to Coral8'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 (e.g.

			nanosecond vs. microsecond), you may get occasional cases of missing or duplicate records.
Timestamp field initial value	TimestampFieldInitValue	Timestamp	If this field is set, then the first time that the query is executed, the adapter will retrieve only records whose timestamp field contains values greater than the value specified in this Initial Value field. (Data values equal to the Initial Value will NOT be retrieved.)

Note that 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 may contain a statement other than SELECT. For example, you may 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 `coral8-services.xml` file), and that 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 *Coral8 Administrator's Guide*.

The poll interval specifies how frequently to read data from the external database server.

You may set property values to specify whether you want the entire table re-read each time, or whether you want to read only records that were 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 will cause 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

Normally, the query specified in the Query property will retrieve all records in the table (or all records that meet 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 since the last query ran. This column must be either a date/time column (e.g. 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 to use, i.e. which column contains the timestamp or counter.

To specify which column to use, you 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, and you must include an ORDER BY clause that refers to the column name that you entered into the Timestamp Field or Counter Field property.

Note that the name of the column that you enter must be the name of the column in the CCL schema, not the name 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 her 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, i.e. which 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" (we'll assume that the column name is the same in the external table and the Coral8 stream). In that case, you would set the adapter's property named Counter Field to "SerialNum", and you would 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 Read 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 that the query executes, then 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, then 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 that you query the table, you will 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.

Here is a complete example.

- The table named Shipments has a column named TimeOfShipment which has a data type compatible with Coral8's TIMESTAMP data type.
- The corresponding column in the stream is named "ShippingTime" and is of course 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 since the year 2000.

To access this table with the Poll From DB adapter, you set the following:

- Timestamp Field: set the adapter property Timestamp Field to the stream column name "ShippingTime".
- Timestamp Field Initial Value: Set the adapter property "Timestamp Field Initial Value" to "1999-12-31 23:59:59.999999").
- The Query's WHERE clause: Set the 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, those 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, i.e. the time at which the trade was execute. 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). Thus the adapter would not see the new value (and would still cache the old value) for that particular stock trade.

Similarly, if a row were deleted (perhaps the purchaser did not have enough money to pay for the stock that he bought), there might not be any indication of that sent to Coral8

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 were deleted. Thus Coral8's local cache of the external table's contents will have **both** the original value and the new value. Therefore, as a general rule, you will probably use incremental retrievals with tables that have only INSERTs, not UDPATEs or DELETEs.

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.

TIP: 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 may be 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 DB* (Read From DataBase) adapter reads information from a table (or view) 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 for processing live data.
This adapter does not support Accelerated Playback.	This adapter supports Accelerated Playback.	This adapter supports Accelerated Playback.
The user specifies the entire database	The user specifies the database table name (or view name) and	The user specifies the name of a file that

query.	the WHERE clause.	contains the data.
The user may 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 "Coral8 Engine Third-Party Software Dependencies" in the *Coral8 Administrator's Guide* for information about supported databases.

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 Coral8 Studio, and properties for this adapter are set using Coral8 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**	**Required?**
DBName	DBName	String	This indicates which database to execute the query on. This is actually be the name of a "service" that is defined in the `coral8-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. This field is	Required

			required.	
Table or View	Table	String	The table from which to retrieve the data. Note that this may be a table or a view.	Required
Where Clause	WhereClause	String	This clause limits the result set. This WHERE clause will be applied on the remote database server, and thus may 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 will be inserted. Note that you should omit the keyword WHERE. E.g. 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, then the default value is 1.	Optional
Rate	Rate	Float	How fast to read the records. This field is optional. If this field is not set,	Optional

			then the timing of the data will be based on the Timestamp field.	
Timestamp Column	TimestampColumn	String	This is the name of the timestamp/datetime column. *Note that this is the name of the field in the external table, not the name of the field in the Coral8 schema.*(This is explained in more detail later in this section.) Note that this column's data type should be compatible with Coral8's TIMESTAMP data type. (See *Datatype Mappings* for tables that show which database data types correspond to Coral8'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 (e.g.	Required

			nanosecond vs. microsecond), you may get occasional cases of missing or duplicate records.	
Timestamp column initial value	TimestampColumnInitValue	Timestamp	If this field is set, then the first time that the query is executed, the adapter will retrieve only records whose timestamp field contains values greater than the value specified in this Initial Value field. (Data values equal to the Initial Value will 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

Note that 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 (which must be configured in the `coral8-services.xml` file), and that 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 Coral8 Administrator's Guide.

The database query will be constructed from the Table Name, the Where clause, and the Timestamp Field. The query will select 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.

The user will also 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 will use the Timestamp field as the ORDER BY clause in the query. Once the rows arrive at the server, the server will treat 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 must be one of the columns of the table; the Timestamp field may 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 will re-run the query. This allows the adapter to avoid buffering old query results, which would significantly hamper performance for large results sets.

Since the query is re-executed each time, and since the table data may change between queries, the data may change between loop iterations.

If you are using looping, you may 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 (e.g. 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 DB* (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. You attach this adapter to a stream by using commands inside Coral8 Studio, and properties for this adapter are set using Coral8 Studio.

The adapter properties are listed in the table below.

Property Name (screen)	**Property Name (Attach Adapter)**	**Type**	**Description**
Query	Query	String	The query (e.g. INSERT statement) that you want to run on the external database server. This field is required. Note that this statement must be written in the language (e.g. SQL) of the external database server, not in CCL. (For example, do not use a CCL Database Statement ("EXECUTE STATEMENT DATABASE...") as the query.)
DBName	DBName	String	Database name. This must be the name of a "service" defined in the `coral8-services.xml` file. The service information will include information about how to connect to the external database server and the name of the database to connect to. This field is required.
Commit frequency	CommitFrequency	Integer	If this field is left empty or is set to 0, Coral8 Server will send a "commit" for each row sent to the external database server. If

			you want to commit every N rows (e.g. every 3rd row) rather than every row, then enter the desired value of N (e.g. 3) into this field. This field is optional. See More Information about Commit Frequency below for more details.

The query is usually an INSERT statement that will insert 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 may use statements other than INSERT. For example, you may use UPDATE or DELETE. You may 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" (stored procedure) 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 `coral8-services.xml` file. For more information about configuring a service to provide access to a database on an external database server, please see the Coral8 Administrator's Guide.

See *Datatype Mappings* for tables that show which database data types correspond to Coral8's data types.

More Information about Commit Frequency. If you set the commit frequency to a value larger than 0, then the adapter will do "commit" operations EITHER when the specified number of rows has been sent, OR i.e. 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 the number of rows to be committed reaches N.

Uncommitted rows will not be committed when a project is stopped. For example, suppose that you set the commit frequency to 5 minutes. As the project runs, it commits writes to the database every 5 minutes. However, when the project is stopped, a final commit is not executed. If the project is stopped 2 minutes after the most recent commit, then the most recent 2 minutes of database writes will not get 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 `coral8-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; i.e. the text of the message that you would like to send.
HTML Mail	HtmlMail	Boolean	Set this to true (i.e. put a checkmark in the box by clicking on the box) if you want the email sent in HTML format. Set this to false (i.e. 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 `coral8-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.

Email: 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. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.

To run this adapter, execute the following command:

```
java com.coral8.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 will look similar to the following:

```
java com.coral8.adapter.JavaEmailAdapter --url= ...
```

Use the command-line parameters shown in the table below. Note that 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	Email address that the email message will show the email as being from.	Required
to	Email address(es) to send the messages to	Required
subject	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	User name	Optional
smtpPassword	Password	Optional
useTLS	Use a TLS secure connection when connecting to the mail server. This option will work only if your mail server supports a TLS connection. (Note that you may use either the "--TLS" option or the "--SSL" option, but not both.)	Optional
useSSL	Use an SSL (Secure Socket Layer) connection when connecting to the mail server. This option will work only if your mail server supports an SSL connection. (Note that you may 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. You 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 -- i.e. 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.coral8.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 (on the same 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 (TLS) 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 Coral8 row/message that arrived at the adapter.

Excel RTD Output Adapter

RTD Refresh Interval

Excel does not update RTD data on every single update. It refreshes RTD data periodically based on a system-wide throttle interval named Excel.Application.RTD.ThrottleInterval. This throttle interval can be set in either of the following ways:

- Via Visual Basic / VBA - in immediate mode or in a startup handler, you can set Application.RTD.ThrottleInterval.
- Via the Registry - On startup, Excel will initialize the throttle interval from the registry value.

  For Excel 2002:

  ```
  HKEY_CURRENT_USER\Software\Microsoft\Office\10.0\
     Excel\Options\RTDThrottleInterval
  ```

  For Excel 2003:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\
   Excel\Options\RTDThrottleInterval
```

The throttle interval can be set to any one of the following values:

ThrottleInterval value	**Effect**
-1	Manual mode. Never automatically refreshes. Excel.Application.RTD.RefreshData must be called.
0	Checks for updates every chance it gets.
> 0	Excel waits at least the specified number of milliseconds before getting updates.

For more information, please refer to the following Microsoft FAQ item from MSDN, which explains the details best:

```
http://msdn2.microsoft.com/en-us/library/
   aa140060(office.10).aspx#odc_xlrtdfaq_howconfigrtdthrottle
```

Ganglia Input Adapter

The *Ganglia* adapter takes data from a Ganglia Monitoring System source and sends the rows to the Coral8 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.

Note that 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.

For more information see the Ganglia Alerts example in the examples/NetworkAndSecurity/GangliaAlerts directory.

E.g. on Microsoft Windows, the directory is typically:

```
C:\Program Files\Coral8\Server\examples\NetworkAndSecurity\GangliaAlerts
```

On UNIX-like operating systems, the directory is typically

```
/home/<username>/coral8/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 will get 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. in the section that describes the JDBC Output adapter.

This adapter is run as an out-of-process adapter.

Set your CLASSPATH environment variable according to the instructions in *Coral8 Java SDK*. (The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.)

Execute the following command:

```
java com.coral8.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.coral8.adapter.JDBCInputAdapter --queryFile=q1.sql ...
```

Note that "--" (two dashes) must precede each parameter name.

Use the command-line parameters shown in the table below. Note that these parameters are case-sensitive.

Parameter	**Description**	**Required?**
queryFile	A file with the SQL query, which may be	Required

	almost any SQL statement that is valid for the database that the adapter is connected to. The query can be parameterized.	
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.	Required
streamUrl	The URL of the stream to send tuples to.	Required
driver	The database driver to use. For example: --driver=sybase.jdbc.driver.SybaseDriver. Coral8 does not bundle JDBC drivers with our product, so customers must provide the drivers themselves. In order for the Coral8 JDBC Input adapter to work, you must include the JDBC driver 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	This must be the name of a file with a tuple descriptor that describes the schema that we 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, then add the retrieved rows to the existing rows. If this is set to false, then replace 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

Note that 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 will initialize its connection to Coral8 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 (e.g. 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.

Tuple Descriptor File

This Tuple Descriptor File must describe the schema of the stream that we are publishing to. This file is required because out-of-process adapters have no way of knowing the schema on Coral8 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.coral8.com/cpx/2004/03/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.coral8.com/cpx/2004/03/cpx_td.x
sd"
      Name="MyTuple"
      >
   <Field Name="field1" xsi:type="StringFieldType" />
   <Field Name="field2" xsi:type="IntegerFieldType" />
</TupleDescriptor>
```

JDBC Output Adapter

A *JDBC Output* adapter allows you to write output to another database server (e.g. an Oracle server) through JDBC.

You must specify several pieces of information, including

- information about connecting to the database server (e.g. 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 will insert a row into the external database. This statement will contain placeholders for parameters; the values for the parameters will be 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 (see JDBC standard documentation for details). The columns in each row are inserted into the query in the order they appear in the row descriptor.

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 Coral8 Server configuration file. (For more information about Coral8 Server configuration file, see the Coral8 Administrator's 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:

Set your CLASSPATH environment variable according to the instructions in *Coral8 Java SDK*. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.

Execute the following command:

```
java com.coral8.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.coral8.adapter.JDBCInputAdapter --queryFile=q1.sql ...
```

Note that "--" (two dashes) must precede each parameter name.

Use the command-line parameters shown in the table below. Note that these parameters are case-sensitive.

Parameter	**Type**	**Description**
queryFile	A file with the SQL query, which may 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.	Required
streamUrl	The URL of the stream to read tuples from.	Required
driver	The database driver to use. Coral8 does not bundle JDBC drivers with our product, so customers must provide the drivers themselves. In order for the Coral8 JDBC Input adapter to work, you must include the JDBC driver 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 db (no tnsnames used):

```
java com.coral8.adapter.JDBCOutputAdapter
--queryFile=/home/lita/QueryFileOutputAdapter --username=lita
--password=CORAL8
--databaseURL=jdbc:sybase:Tds:<host>:<port>/<database>--
streamUrl=ccl://pc03.c8.com:7777/Stream/DbWrite/OutStream
```

When a value from the Coral8 output data stream is inserted into an external database, the data type of the Coral8 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. Coral8 does not support YEAR TO MONTH intervals.

Your query must use the same number of binding variables as you have 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 simply 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 that the adapter can attach to them.

To shut down the adapter, send an interrupt signal (e.g. ctrl-C).

Connectivity Instructions for UNIX-like Operating Systems

Here's a sample command to call JDBCOutputAdapter to connect to a remote database server:

```
java com.coral8.adapter.JDBCOutputAdapter
 --queryFile=/home/lita/QueryFileOutputAdapter
 --username=lita --password=CORAL8
 --databaseURL=jdbc:sybase:Tds:<host>:<port>/<database>
 --driver=sybase.jdbcc.driver.SybaseDriver
 --streamUrl=ccl://pc03.c8.com:7777/Stream/OutStream
```

For more information about connecting on UNIX-like operating systems, see Connectivity Instructions for UNIX-like Operating Systems.

JMS Input Adapter

This adapter has been deprecated. Use the JMS Adapter instead.

This adapter attempts to establish a JMS (Java Messaging Service) session and subscribes to the given topic, then listens on that session and sends incoming messages as tuples into the 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.

The non-GD JMS input and output adapters work with JMS messages that have message body type "MapMessage" or "TextMessage".

The JMS Input adapter is an out-of-process adapter.

To run it, execute the following steps:

Set your CLASSPATH environment variable according to the instructions in *Coral8 Java SDK*. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.

Update your CLASSPATH to include the necessary JMS libraries. (Coral8 does not supply the libraries for any JMS implementation.)

Execute the following command:

```
java com.coral8.adapter.JMSInputAdapter <parameters>
```

Each parameter is of the form

--<paramname>=<paramvalue>

Note that "--" (two dashes) must precede each parameter name. For example:

```
java com.coral8.adapter.JMSInputAdapter --topic=...
```

The command-line parameters are specified below. Note that these parameters are case-sensitive.

Parameter	**Description**	**Required?**
Topic	The name of the topic to publish/subscribe to.	Required
Factory	Connection factory name	Required
factoryClass	Factory class	Optional
url	The CCL URL of the stream to write to.	Required
Host	Host name: The host on which the WebLogic server (not Coral8 Server) is running.	Optional
Port	port: The port number of the Weblogic server (not Coral8 Server).	Optional
c8.baseHostPort	Host and port of Coral8 Server. See below for details	Optional

Note that 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.

c8.baseHostPort

This adapter allows the user to set the Sytem Property c8.baseHostPort.

The c8.baseHostPort should look like:

host:port

where host is the host name of the computer that is running Coral8 Server to which the adapter will connect, and the port is the port number of Coral8 Server to which the adapter will connect.

The adapter needs this host and port information to assemble a URL to contact the Manager. (The adapter calls the Manager to resolve the CCL url that the adapter is given as a command line parameter.) To set the property, use the java -D option. For example:

```
java -Dc8.baseHostPort=pc03:6790
 com.coral8.adapter.JMSInputAdapter
 --topic=weblogic.jws.jms.MyJMSTopic
```

```
--factoryName=weblogic.jws.jms.TopicConnectionFactory
--factoryClass=weblogic.jndi.WLInitialContextFactory
--url=ccl://pc03.c8.com:6790/TestJMS/instream2
--host=localhost
```

If you do not specify the c8.baseHostPort, the adapter will assume the default values for host and port: host name: localhost port number: 6789

Note: Do not confuse the c8.baseHostPort with the "host" and "port" parameters. Those host and port parameters are specific to the WebLogic JMS Server and they specify where the WebLogic server is running (as opposed to where the Coral8 Server is running).

JMS Output Adapter

This adapter has been deprecated. Use the JMS Adapter instead.

This adapter attempts to look up a JMS (Java Messaging Service) topic and publish to it the data received from the Coral8 Server. Messages sent out are JMS MapMessages. The order and type of fields must exactly match the tuple descriptor.

The non-GD JMS input and output adapters work with JMS messages that have message body type "MapMessage" or "TextMessage".

The JMS Output adapter is an out-of-process adapter.

To run it, execute the following steps:

Set your CLASSPATH environment variable according to the instructions in *Coral8 Java SDK*. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.

Update your CLASSPATH to include the necessary JMS libraries. (Coral8 does not supply the libraries for any JMS implementation.)

Execute the following command:

```
java com.coral8.adapter.JMSOutputAdapter <parameters>
```

Note that "--" (two dashes) must precede each parameter name. For example:

```
java com.coral8.adapter.JMSOutputAdapter --topic=...
```

The command-line parameters are specified below. Note that these parameters are case-sensitive.

Parameter	**Description**	**Required?**
Topic	The name of the topic to publish/subscribe to.	Required
factoryName	Connection factory name	Required

factoryClass	Factory class	Optional
url	The CCL URL of the stream to write to.	Required
messageType	May be either "MapMessage" or "TextMessage". The default is "MapMessage". If you use "TextMessage", the adapter will accept XML data in the first data field and wrap it in a TextMessage object; all fields after the first field will be ignored. "TextMessage" is compliant with the JMS specification.	Optional
Host	Host name: The host on which the WebLogic server (not Coral8 Server) is running.	Optional
Port	Port number of the WebLogic server (not Coral8 Server).	Optional
c8.baseHostPort	Host and port of Coral8 Server. See below for details.	Optional

c8.baseHostPort

This adapter allows the user to set the System Property c8.baseHostPort.

The c8.baseHostPort should look like:

host:port

where host is the host name of the computer that is running Coral8 Server to which the adapter will connect. the port is the port number of Coral8 Server to which the adapter will connect.

The adapter needs this host and port information to assemble a URL to contact the Manager. (The adapter calls the Manager to resolve the CCL url that the adapter is given as a command line parameter.) To set the property, use the java -D option. For example:

```
java -Dc8.baseHostPort=pc03p:6790
com.coral8.adapter.JMSOutputAdapter
 --topic=weblogic.jws.jms.MyJMSTopic
 --factoryName=weblogic.jws.jms.TopicConnectionFactory
 --factoryClass=weblogic.jndi.WLInitialContextFactory
 --url=ccl://localhost:6789/Stream/Default/TestJMS/instream1
 --host=localhost
```

If you do not specify the c8.baseHostPort, the adapter will assume the default values for host and port: host name: localhost port number: 6789

Note: Do not confuse the c8.baseHostPort with the "host" and "port" parameters. Those host and port parameters are specific to the WebLogic JMS Server and they specify where the WebLogic server is running (as opposed to where the Coral8 Server is running).

JMS Adapter

This adapter can be configured as either an input adapter or an output adapter.

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, then 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 Coral8 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, then it takes data received from a Coral8 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 Coral8 stream's tuple descriptor.

The JMS server should use the JMS 1.1 protocol.

This adapter may be configured to use Coral8's Guaranteed Delivery (GD) or "normal" (non-GD) delivery when reading from or writing to a Coral8 stream.

The non-GD JMS input and output adapters work with JMS messages that have message body type "MapMessage" or "TextMessage".

If GD is turned on, then the adapter provides "At Least Once" semantics for outbound MapMessage and TextMessage messages from Coral8 and "Exactly Once" semantics for inbound messages.

The JMS adapter is an out-of-process adapter.

To run it, 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. (Note that 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 Coral8 Java SDK. The instructions for setting CLASSPATH apply to all out-of-process Java adapters, whether provided by Coral8 or written by the customer.
5. Update your CLASSPATH to include the necessary JMS libraries. (Coral8 does not supply the libraries for any JMS implementation.)
6. Execute the following command:

   ```
   java com/coral8/adapter/JMSAdapter <parameters>
   ```

   Each parameter is of the form

   --<paramname>=<paramvalue>

   Note that "--" (two dashes) must precede each parameter name. For example:

   ```
   java com/coral8/adapter/JMSAdapter --topic=test1 --GD=true ...
   ```

   The command-line parameters are specified in a table later in this section. Note that these parameters are case-sensitive.

Parameter	**Description**	**Required?**
kind={input | output}	Indicates whether the adapter should run as an input adapter or as an output adapter.	Yes
serverURL=<URL to server>	URL of Coral8 Server, for example `http://localhost:6789`. If your manager and containers are separate, this must be the URL of the manager.	Yes
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}	Use Guaranteed Delivery. The default value is false. *Note: for best results, the stream's GD setting should match the adapter's setting.*	No
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>	Name of the factory in JMS. This varies depending upon the which vendor's JMS you are using.	Yes
initialNamingFactory=<factoryClass>	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.	No
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`	Yes
messageType=<message type>	"MapMessage" or "TextMessage". The default is "MapMessage".	No
C8username=<userID>	If this is set, then the connection to Coral8 Server uses credentials to authenticate the user.	No
C8password=<password>	Used in the credentials, along with the C8username.	No
JMShost=<hostname>	The host computer on which the JMS server is running. The default value is `localhost`.	No
JMSport=<port>	The port that the JMS server is using. The default value is 16400.	No
batchSize=<integer>	If the adapter is using GD, then the adapter will attempt to send rows in batches of the specified size. • Input: When running as an input adapter, the adapter will send the data to the Coral8 Server as soon as either the batch size or commitIntervalMsec is reached, whichever comes first. Note: if both batchsize and rate are set, rate is used. • When running as an output adapter, the adapter will do a JMS transaction when it reaches either the batchSize or the commitIntervalMsec, whichever comes first. The default batch size is 500.	No

rate=<float>	For GD input, attempts to send at approximately this rate. If both rate and batchSize are set, rate is used.	No
commitIntervalMsec=<integer>	Send batch at least this often (in milliseconds) unless there are no rows to send. The default value is 500 (milliseconds).	No
statusFile=<filename>	The file name to which the adapter should write status if the adapter is either • a GD output adapter, or • a GD input adapter for topics. File names MUST be different for each instance of the adapter. The default file name is `JMSAdapter.status`	No
debug=true	Print out debug information about "high-level" operations. By default, debugging is off.	No
debugVerbose=true	Print out debug information about each message. By default verbose debugging is off.	No

Although the GD (Guaranteed Delivery) settings of the stream and adapter are not required to match, Coral8 strongly recommends that the settings match -- in other words 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 may 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 skewed 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.

Log File Reader Adapter

Installation

This section describes how to install this adapter. (Unlike most other Coral8-supplied adapters, this adapter is not installed automatically.)

On UNIX-like operating systems, this adapter is provided in the form of a .tgz file named `c8-clicks-adapter.tgz`. Source code and an example .properties file are included. You can find the .tgz file in the `adapters` directory, which is typically

```
/home/<userID>/coral8/server/adapters
```

When you extract the files from the .tgz file, you should put those files into the `/etc/c8clicks` directory. Typically, you will use shell commands similar to the following:

```
mkdir /etc/c8clicks
cp c8-clicks-adapter.tgz /etc/c8clicks
# De-compress the .tgz file by using the gnu "unzip" utility.
gunzip c8-clicks-adapter.tgz
# Extract the files.
tar -xvf c8-clicks-adapter.tar
```

Random Tuples Generator Adapter

The *Random Tuples Generator* adapter (Sometimes called the *Random Messages Generator)* generates random rows (tuples) according to the given schema and sends the rows to the stream. This adapter is useful primarily for prototyping and basic testing.

Property Name (screen)	**Property Name (Attach Adapter)**	**Type**	**Description**
Rate	Rate	Float MIN: 0 MAX: 1000000 Default: 100	The number of Rows generated per second. It is important to note that this Rate may not exceed the "MaxRate" parameter set in the server configuration file.
Row Count	RowCount	Long MIN: 0 MAX: 2000000000 DEFAULT: 0 (infinity)	How many rows to generate. If the row count is 0, the value is treated as infinite.

CAUTION: Data generated by this adapter is not evenly distributed across the range of possible values for each data type. The table below shows the range of value generated for each data type. Note that even within this range, values are not necessarily evenly distributed.

Data Type	**Range of Values**

C8Bool	true/false
C8Int	0 .. 99 inclusive
C8Long	0 .. 99 inclusive
C8Float	0.0 .. 10.0 exclusive (should never get 10.0)
C8Interval	0 .. 9 inclusive
C8Timestamp	Sets value to current time
C8CharPtr	2 characters from the following ranges a..z, A..Z, 0..9
C8BlobPtr	2 bytes each with range 0..255

Regular Expressions: Read From File Using a Regular Expression Adapter.

Read From File Using a Regular Expression adapter reads an input file line by line, matches each line against a given regular expression (which may contain multiple subexpressions), 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, suppose that you search using the following regular expression:

```
/(J).*(SMITH)/
```

and that the input rows are:

```
1) JOHN SMITH
2) JANE SMITHERS
3) JANE AUSTEN
4) DAVE SMITH
5) JANET SMITHSONIAN
```

This will return 3 rows (taken from rows 1, 2, and 5 above), each with 2 columns of output:

```
J SMITH
J SMITH
J SMITH
```

More sophisticated patterns would allow you to extract a broader range of values (e.g. the complete last names).

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.

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. 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 greater than 1, then after finishing reading the file, 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, then repeat forever.
Rate	Rate	Float	If this property is non-zero, then the adapter will read data from file with the given rate (per second). Any timestamps in the input file will be ignored.
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 will be sent 5 seconds (5000000 microseconds) after the module starts. If blank, the first row will be 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 sub-expressions.
Ignore Mismatch	IgnoreMismatch	Boolean	If true, a line not matching the regular expression is ignored. If false, then if there is a line that does not match the regular expression, the adapter raises an error condition and stops.
Log Mismatch	LogMismatch	Boolean	If true, lines that don't match the regular expression are logged. Otherwise, the mismatched lines are dropped silently. (This option is only used when Ignore Mismatch option is set to true.)
Timestamp column format	TimestampColumnFormat	String	The Timestamp column format specifies the format of the timestamp columns (e.g. 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. For more information, see Reading, Writing, and Converting Timestamps. Note that this format specifier applies to all input columns of type TIMESTAMP. (In this adapter, the Timestamp column format does NOT apply 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.

Note that 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 Coral8 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 Coral8 Server running on the same computer as Coral8 Studio, or as specified in the preferences for Coral8 Studio. For more information, see Setting The Base Folder For File Input/Output Adapters. 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 Coral8 Server. The Base Folder setting for the Server is specified during the installation process, and may also be changed later in the Server's `coral8-server.conf` file. The base folder setting for Coral8 Studio may be set from the Tools->Settings command on the Studio menu.

The "Edit" button for the filename property. The edit button opens an editor that will allow you to edit the 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", then Studio will open 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 will be the one specified by the operating system's file associations, and on UNIX-like operating systems Coral8 Studio will open the editor specified by the EDITOR environment variable.

When you click the Edit button, Coral8 Studio will look for the file in the Coral8 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 attempts to open a TCP connection to a given address (specified through Host and Port properties) and, once connection is established, 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 "Read From File Using a Regular Expression" Adapter 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	Host name or IP address of the data source
Port	Port	Integer Min: 1 Max: 65535	Port number of the data source
Regular expression	RegEx	String	Regular expression to use when parsing rows.
Fields	Fields	String	Comma-separated list of fields corresponding to sub-expressions.
Timestamp column format	TimestampColumnFormat	String	The Timestamp column format specifies the format of the timestamp columns (e.g. 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. For more information, see Reading, Writing, and Converting Timestamps. Note that this format specifier applies to all input columns of type TIMESTAMP. (In this adapter, the Timestamp column format does NOT apply 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.
Ignore Mismatch	IgnoreMismatch	Boolean	If true, a line not matching the regular expression is ignored. If false, error condition is raised and adapter is stopped.
Log Mismatch	LogMismatch	Boolean	If set to true, lines that don't match the regular expression are logged. Otherwise, the mismatched lines are dropped silently. (This option is only used when IgnoreMismatch option is set to true.)

Note that 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 attempts to open 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. This field is required.
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)	Refresh interval (how often to poll the RSS feed at the given URL). This may be a number of microseconds or may be a string of the form "# <unit>", e.g. "10 seconds" or "1 minute". Note that not all valid format for the INTERVAL data type may be used in this field.

Note that each input stream has a property (see the stream's Properties tab in Coral8 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 Coral8 schema column names and RSS feed "item" elements. For example, if the Coral8 schema has 3 fields named "title", "link" and "description" then the Coral8 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.html#item573</guid>
</item>
```

will be

```
"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, you may do so by:

Copying and modifying the files c8_rss_reader.cpp and c8_rss_reader.adl;

Compiling the .cpp file into a shared object library (.so) or dynamic link library (.dll).

Copying 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 *Coral8 C/C++ SDK*.

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 allows 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. Note that the Coral8 SNMP Get and Set adapters support only SNMP versions 1 and 2c.	Optional
Community String	CommunityString	String	The SNMP password. The default value is "public". WARNING: The SNMP community string will be transmitted unencrypted!	Optional
Host	Host	String	Host machine. Either an	Optional

machine			IP address or a symbolic address may be used. Coral8 will not address issues of firewalls in accessing the host machine. If the host machine is not supplied, it will default to localhost.	
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 the responsibility of the user to ensure the SNMP datatype associated with the OID matches the Coral8 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 more real-life situations.	Optional

Note that 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) the data will be sent to or from. The schema of the stream will depend upon the device that you are getting information for. To determine the fields needed in the stream schema, you may need to look at the MIB (Management Information Base) that describes the device.

This Coral8 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.

Coral8 Datatypes vs. SNMP Datatypes

SNMP supports data types that do not directly correspond to Coral8 data types. For example, Coral8 does not have unsigned data types. Coral8 also does not support the SNMP Sequence and Sequence-of datatypes without user assistance.

Coral8 maps the SNMP datatypes as follows:

ASN.1 Datatype	Coral8 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_OPAQUE_FLOAT	C8_FLOAT
ASN_OPAQUE_DOUBLE	C8_FLOAT

Users should provide a fixed schema that takes the entire SNMP request and sends a single message to the Coral8 engine. This single message will provide a time basis for the SNMP query.

There are several items you must take into consideration.

- You should define columns that will hold SNMP type Counter32 as Long rather than Integer to allow proper extraction. Otherwise, overflow may result in negative values.
- Coral8 has only signed integer and long values while ASN.1/net-snmp has both signed and unsigned values.

- Coral8 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 Coral8 uses microseconds. This adapter will multiply 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 will generally be 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 allows you to send SNMP information. The Simple Network Management Protocol (SNMP) is an Internet-standard protocol for managing devices on IP networks.

When the Coral8 engine sends a message to the Set adapter, the only column present is a string.

A PDU (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. Note that the Coral8 SNMP Get and Set adapters support only SNMP versions 1 and 2c.	Optional
Community String	CommunityString	String	The SNMP password. WARNING: The SNMP community string will be transmitted unencrypted!	Optional
SNMP Host machine	Host	String	Host machine. Either an IP address or a symbolic address may be used.	Optional

			Coral8 will not solve issues of firewalls in accessing the host machine.	
OIDs	OIDList	String	The Object ID to which messages will be sent.	Optional

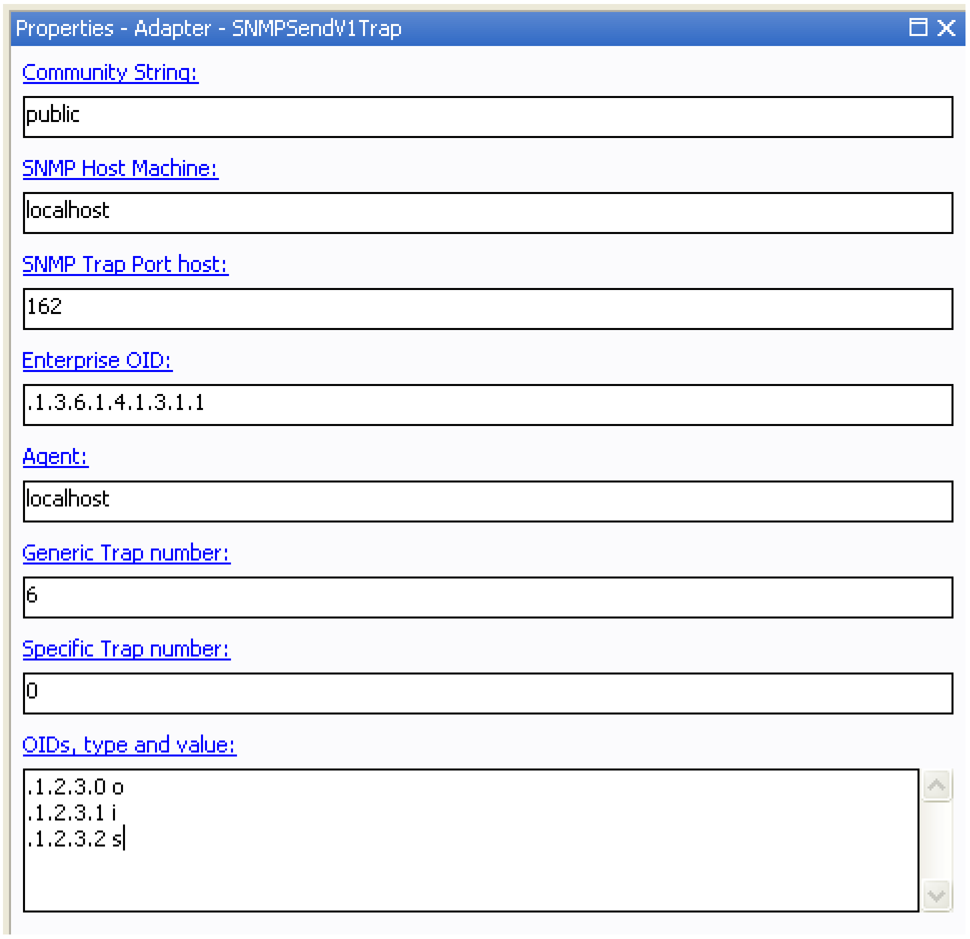
These properties determine which SNMP object the data will be sent to or from. The schema of the stream will depend upon the device that you are getting information for. To determine the fields needed in the stream schema, you may need to look at the MIB (Management Information Base) that describes the device.

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.

For information about mapping Coral8 data types to SNMP data types, please see Coral8 Datatypes vs. SNMP Datatypes.

SNMP Send V1 Traps Adapter

The Coral8 engine supports sending V1 traps and V2c notifications. The adapter's Properties View for V1 traps appears as:


Properties - Adapter - SNMPSendV1Trap

Community String:

public

SNMP Host Machine:

localhost

SNMP Trap Port host:

162

Enterprise OID:

.1.3.6.1.4.1.3.1.1

Agent:

localhost

Generic Trap number:

6

Specific Trap number:

0

OIDs, type and value:

```
.1.2.3.0 o
.1.2.3.1 i
.1.2.3.2 s
```

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 should represent a decent approximation to the uptime of the system hosting Coral8 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 will map 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	Object ID
s, x, d	Octet string
n	Null
b	bit string
U	ASN_OpaqueU64
I	ASN_OpaqueI64
F	ASN_Opaque_Float
D	ASN_Opaque_Double

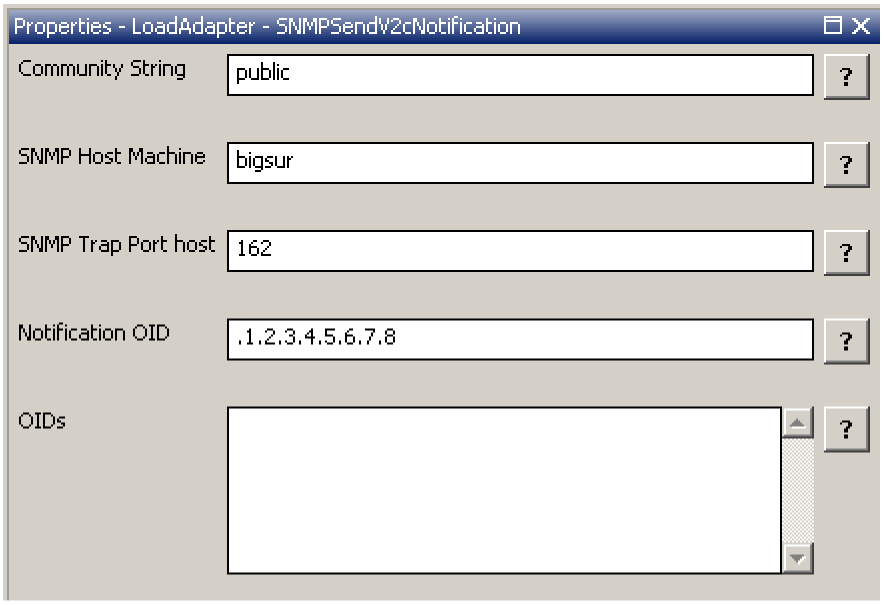
Which datatype should be used? This depends on the datatype on the host machine of the OID. If numeric datatypes are used, it is not necessary to provide a valid MIB OID, a made-up OID as illustrated in the screenshot will suffice. A made-up OID places the burden of processing on the trap daemon on the host machine. This can be convenient as this corresponds closely to a "name value" pair on the host machine.

Using a symbolic OID requires a proper correspondence of datatypes.

The Coral8 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 Coral8 engine supports sending V1 traps and V2c notifications. The adapter Properties View for V2c Notifications looks like the following:


Properties - LoadAdapter - SNMPSendV2cNotification

Community String	public
SNMP Host Machine	bigsur
SNMP Trap Port host	162
Notification OID	.1.2.3.4.5.6.7.8
OIDs	

The use of V2c notifications is even simpler than V1 traps. 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 Coral8 matching of schema values to Net-snmp values. See the table that describes Net-SNMP trap datatypes.

Sybase RAP Output Adapter

The Sybase RAP adapter reads a Coral8 data stream and transforms it into a feed handler for Sybase RAP. This out-of-process C/C++ adapter, named **c8_sybase_rap_adapter**, is installed in the **adapters** directory under your Coral8 Server or Coral8 Studio installation directory. Note that the adapter is not available for all platforms. Check the release notes for the currently supported platforms.

The Sybase 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 purpose of each preference. Note that you must be familiar with Sybase RAP in general and your Sybase RAP installation in particular, in order to set up your preferences file correctly. 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 Coral8 Server or Coral8 Studio installation directory.

To run the adapter, simply execute the following command, replacing *my_prefs_file.xml* with the name of your preferences file:

```
c8_sybase_rap_adapter my_prefs_file.xml
```

Windows Event Logger Adapter

Schema

The Windows Event Log adapter will send messages to the Coral8 Server in the following schema:

Field	Type	Description
Category	String	Text associated with the Category Number
CategoryNumber	Integer	Category of 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 user responsible for event
LogName	String	Event Log in which the event was generated
Site	String	Text name of the ISite of the event

The event schema may also be found in the `wineventlog.ccs` file. If you installed Coral8 Engine to the default directory, this file will be in the `C:\Program Files\Coral8\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.

XML: Read From XML File

A *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 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 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. 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 greater than 1, then 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, then repeat forever.
Rate	Rate	Float Min: 0.0000001 (if set to a	If this property is non-zero, then the adapter will read data from

		number greater than zero)	the input file at the specified rate (per second). Any row timestamps in the input file will be ignored (i.e. will not affect the rate at which data is read and sent to the server). The default is to leave this unset, in which case the row timestamp controls 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 will be sent 5 seconds (5000000 microseconds) after the module starts. If blank, the first row will be sent immediately after the module starts.
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 that 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 2 rows is shown below:

```
<Tuple xmlns="http://www.coral8.com/cpx/2004/03/"
 Timestamp="946713600000001">
   <Field Name="ID">1</Field>
   <Field Name="BandName">Soft White Underbelly</Field>
</Tuple>
<Tuple xmlns="http://www.coral8.com/cpx/2004/03/"
 Timestamp="946713600000002">
   <Field Name="ID">2</Field>
   <Field Name="BandName">The Quarrymen</Field>
</Tuple>
```

The "http://www.coral8.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. The row timestamp follows the usual rules for Coral8 timestamps; i.e. it is the number of microseconds since midnight January 1, 1970.

Lines 2-3 contain the field values for one row and lines 6-7 contain the field values for a separate row.

Note that 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.

Note also that the XML file is not necessarily formatted as shown in the example above; the usual format is spread out less and is less "human-readable".

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.

Format Of TIMESTAMP Values In The File. 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.

Format Of INTERVAL Values In The File. 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]]] (e.g. 1 02:03:04.000005)
- [D day[s]][ ][HH hour[s]][ ][MI minute[s]][ ][SS[.FF] second[s]] (e.g. 1 day 2 hours 3 minutes 4.000005 seconds)

For a complete list of valid formats for INTERVAL values, search for 'INTERVAL Literals" in the CCL Reference.

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 "Browse" button for the filename property. The Adapter Properties screen in Coral8 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 Coral8 Server running on the same computer as Coral8 Studio, or as specified in the preferences for Coral8 Studio. For more information, see Setting The Base Folder For File Input/Output Adapters. 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 Coral8 Server. The Base Folder setting for the Server is specified during the installation process, and may also be changed later in the Server's `coral8-server.conf` file. The base folder setting for Coral8 Studio may be set from the Tools->Settings command on the Studio menu.

The "Edit" button for the filename property. The edit button opens an editor that will allow you to edit the 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", then Studio will open 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 will be the one specified by the operating system's file associations, and on UNIX-like operating systems Coral8 Studio will open the editor specified by the EDITOR environment variable.

When you click the Edit button, Coral8 Studio will look for the file in the Coral8 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.

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. 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. 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 then 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, then output is written to a single file, which may 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. 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.
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. Note that when Coral8 Server writes INTERVAL values in XML format, the values are always written as a 64-bit signed integer representing a number of microseconds.

The "Browse" button for the filename property. The Adapter Properties screen in Coral8 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 Coral8 Server running on the same computer as Coral8 Studio, or as specified in the preferences for Coral8 Studio. For more information, see Setting The Base Folder For File Input/Output Adapters. 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 Coral8 Server. The Base Folder setting for the Server is specified during the installation process, and may also be changed later in the Server's `coral8-server.conf` file. The base folder setting for Coral8 Studio may be set from the Tools->Settings command on the Studio menu.

The "Edit" button for the filename property. The edit button brings up an editor that will allow you to view the file whose name you entered into the filename field. If the file's extension is "csv" or "xml", then Studio will bring 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 will be the one specified by the operating system's file associations, and on UNIX-like operating systems Studio will bring up the editor specified by the EDITOR environment variable.

When you click the Edit button, Coral8 Studio will look for the file in the Coral8 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 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 that 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. (This property is required.)
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 following (this example is based on raw data from Coral8 that consists of 2 fields per message, "Symbol" and "Price"):

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:c8=http://www.coral8.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 (e.g. for custom HTTP headers), you may copy and modify the adapter code in the file

```
.../coral8/server/sdk/c/examples/c8_write_xml_over_http.cpp
```

To create a customized version of this adapter, do 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, and the .adl file into the server and Studio plugins directories.

The RSS adapter 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 Coral8 C/C++ SDK.

SNMP Adapter Information

This section provides information about SNMP (Simple Network Management Protocol) adapters.

Motivation

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 Coral8 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.

Coral8 allows you to both get and set SNMP OIDs.

Configuring Your Environment for SNMP Adapters

The Coral8 distribution assumes there is 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 Coral8 distribution. In order to preserve existing SNMP configuration files, environment variables are used to append the Coral8 Server information to the user 's configuration. Care has been taken to not overwrite user configuration files, but, as in all similar installation processes, users should back up their 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 Coral8 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. The user is responsible for this and all other Net-SNMP installation configurations.

In the following list, the CORAL8_INSTALL is the directory in which Coral8 Server is installed. The default locations of Net-SNMP files and directories are assumed. If the user sets an environment variable, Coral8 software will not overwrite that variable.

- `INSTALL_BASE=CORAL8_INSTALL/snmp`

  This is the snmp directory in the installation directory of Coral8 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 CORAL8_INSTALL/snmp, user configuration and MIB files will not be overwritten.

- `NETSNMP_DEFAULT_MIBS=C:/usr/share/snmp/mibs; D:/usr/share/snmp/mibs; CORAL8_INSTALL/mibs`

  This is the location of the MIBs directory. The Net-snmp defaults are first. This is because users are expected to maintain their own MIBs. The Coral8 MIBs cannot cover all user 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=CORAL8_INSTALL/snmp`

  This directory is used only by the snmpd daemon and is not expected to be used in Coral8 Server environment. This corresponds to the SNMP_PERSISTENT_DIR variable.
- SNMPLIBPATH=CORAL8_INSTALL/snmp

  This is the location of the net-snmp libraries. Overriding this variable should be performed with caution if the user version of net-snmp is not Net-snmp 5.4. This should probably be left alone.
- `SNMPSHAREPATH=CORAL8_INSTALL/snmp`

  The shared path. On linux this is `/usr/local/share/snmp`
- `SNMPCONFPATH=CORAL8_INSTALL/snmp`

  A list of directories fo search for configuration files. Refer to http://net-snmp.sourceforge.net/docs/man/snmp_config.htm for more information.
- `SNMPDLMODPATH=CORAL8_INSTALL/snmp`

  The dynamically linked library path. See http://net-snmp.sourceforge.net/docs/man/snmp_config.html.
- `NETSNMP_TEMP_FILE_PATTERN=CORAL8_INSTALL/snmp/snmpdXXXXXX`

  The temp file pattern for the mktemp() function.

Configuring the coral8-server.conf File for SNMP

SNMP adapters usually need to reference information in MIBs (Management Information Bases). You can set a preference in the `coral8-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 `coral8-server.conf` relevant to these preferences is in the "Coral8/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\Coral8\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.

Coral8 Drivers

This chapter describes the custom drivers (such as database drivers) provided by Coral8. Note that many major database vendors, such as MySQL, supply their own drivers, so Coral8 does not supply drivers for them.

Configuring Coral8 Drivers

To use a Coral8 driver to communicate with an outside entity, such as a database server, you must configure the **coral8-services.xml** file. Please see the *Coral8 Administrator's Guide* for instructions.

Status Information

This appendix provides information about Message Groups and Message Names that were described in the section titled Monitoring Servers and Projects.

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 every ~1 sec (i.e. 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 every ~1 sec	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	How many

		every ~1 sec	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 every ~1 sec	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 every ~1 sec	For how long the application is running (available only if application is running).
RpcFailedMessages	Long	Updated every ~1 sec	The total number of failed RPC calls for the project.
RpcSentMessages	Long	Updated every ~1 sec	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

Description CclCompilerInfo status messages contain information about Coral8 Compiler options used when the given CCL Application was compiled into CCX Program.

ObjectID The ObjectID is the full path to the given CCL Application in the form:

`<WorkspaceName>/<ApplicationName>` Where `ApplicationName` is the project's (i.e. 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 Coral8 Repository specified by RepositoryPath.
CcxFile	String	Status object only, never updated.	The path to the CCX file relative to Coral8 Repository specified by RepositoryPath.
CclName	String	Status object only, never updated.	The "load name" for the top module (i.e. "application name").
Other Compiler command line parameters (see compiler help for details).	String	Status object only, never updated.	The compiler parameters used.
RepositoryPath	String	Status object only, never updated.	The path to the Coral8 Repository. The repository path name is the root for relative pathnames, e.g. for the .ccp and .ccl files (projects and modules).

Status Information for a CCL Query

Description

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 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 every ~1 second.	The total number of failed RPC calls for the given CCL query.
RpcSentMessages	Long	Everywhere, updated every ~1 second.	The total number of successful RPC calls for the given CCL query.
OutputMessages	Long	Everywhere, updated 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

Description CclStreamPairInfo status messages contain information about Coral8 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>/<Strea
mName>
```

Where `ApplicationName` is the project's (i.e. the top module's) load name.

*ObjectID*2 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>/<Strea
mName>
```

Where `ApplicationName` is the project's (i.e. 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 1 time per second.	The value is the average latency between ObjectID1(stream1) and ObjectID2 (stream2).

Status Information about a Workspace

Description 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	The Manager's

		only, never updated.	version string.

Status Information about a Container

Description ContainerInfo status messages contain runtime information about the specified Container.

ObjectID 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.
ContainerAdded	String	Status Stream Only, updated on event	Event: Container added (value = "Active" or "Passive")
ContainerKilled	String	Status Stream Only, updated on event	Event: Killing container (value = reason)
ContainerRemoved	String	Status Stream Only, updated on event	Event: Container removed (value = reason)
CPUTime	Interval	Everywhere, updated every ~1 sec	CPU Time used by Coral8 Server process since start
CPUUtilization	Float	Everywhere, updated every ~1 sec	Percentage of CPU utilized by the Coral8 Server process (1 = 100%)
HeartbeatPeriod	Interval	Status object only, never updated.	How often the container sends heartbeats.

LoadLimit	Long	Status object only, never updated.	Container load limit (max number of CCX modules allowed to run)
LogErrorCount	Long	Everywhere, updated every ~1 second.	The total number of Errors that have been logged in the server log for a given container. This number is reset to 0 if the container is restarted.
LogWarningCount	Long	Everywhere, updated every ~1 second.	The total number of Warnings that have been logged in the server log for a given container. This number is reset to 0 if the container is restarted.
ManagerURI	String	Status object only, never updated.	The Manager's URI this Container is connected to.
StartTime	Timestamp	Status object only, never updated.	When the Container was started.
State	String	Status object only, never updated.	Container state (Active or Passive). Note: Only available through *manager's* GetContainerStatus and GetManagerStatus calls
TotalMemory	Long (bytes)	Everywhere, updated every ~1 second	Total memory available to Coral8 Server (bytes)
UsedMemory	Long (bytes)	Everywhere, updated every ~1 second	Total memory used by Coral8 Server

			(Bytes)
Version	String	Status object only, never updated.	The Container's version string.

Status Information about a Manager

Description 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	String	Manager	Event: manager

ParticipatingInElection		status stream only, updated on event	HA participating in primary elections (value = reason)

Daylight Saving Time and the Coral8 Time Zone Database

This appendix describes how to configure the Coral8 Engine to deal with time zone issues, including:

- Change the start and end dates of Daylight Saving Time.
- Create custom time zones.
- Customize information (such as time zone abbreviations) for existing time zones.

Background

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. For all Coral8 purposes, GMT and UTC are equivalent. For details about GMT and UTC, see the U.S. Naval Observatory web page: http://aa.usno.navy.mil/faq/docs/UT.html.) 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.

Internally, Coral8 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 (e.g. to a string in a form such as "YYYY-MM-DD HH24:MI:SS.FF"), Coral8 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, Coral8 also takes into account the local time zone and GMT. To allow customers to specify information such as whether their local time zone uses daylight savings time, Coral8 provides a time zone "database" that customers can customize. Customers 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 Coral8.
- A desire to use different abbreviations (e.g. "PDT") for certain time zones, e.g. to avoid duplicate abbreviations. (This is discussed in more detail later.)

Other customizations are also possible.

Coral8's time zone "database" is in the form of a Comma-Separated Values (CSV) file named `c8_timezones.csv`, which contains one row per time zone. The Coral8 Server and Coral8 Studio each have a copy of this file in their respective plugins directories, e.g. on Microsoft Windows

```
C:\Program Files\Coral8\Server\plugins\c8_timezones.csv
C:\Program Files\Coral8\Studio\plugins\c8_timezones.csv
```

and on UNIX-like operating systems

```
/home/<userid>/coral8/server/plugins/c8_timezones.csv
/home/<userid>/coral8/studio/plugins/c8_timezones.csv
```

The reason that the file is located in both the 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 will be in local time.

Users may specify timezone information in various CCL function calls (e.g. TO_TIMESTAMP()) and in some Coral8 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 the user has not changed the default abbreviations in the Coral8 time zone database).

Users must run Coral8 software on machines with proper time zone configurations.

Daylight Savings Time

If the user specifies a particular time zone, and if that time zone uses daylight savings time, then Coral8 software takes that into account. Specifically, Coral8 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 be taken into account if the local time zone uses daylight savings time and if 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 Coral8 Time Zone Database to take into account the new start and end dates.

The United States changed its start and end dates for daylight savings time; the change takes effect in 2007. If you are using Coral8 Engine 4.6.2 or earlier and use U.S. time zones in any data, you should check and, if necessary, update the Coral8 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, since these undefined and/or ambiguous times may still be input, the Coral8 software must 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. Coral8 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, Coral8 will interpret 02:00 PDT as 01:00 PST on this morning.

Users truly concerned with these matters should either not use daylight savings time or should use GMT times (which do not include daylight savings time).

Duplicate Time Zone Abbreviations

Although you might expect that there would be only 24 time zones on earth, there are many more. Some countries, such as India, have time zones that are not a multiple of an hour different from GMT. The Coral8 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 "EST" ("Eastern Standard Time") to refer to the time zone on the eastern side of their respective countries. Since each time zone abbreviation in the Coral8 time zone database must be unique, Coral8 users in the U.S. may prefer to keep "EST" as a reference to U.S. Eastern Standard Time and use a different abbreviation (e.g. "AEST") for Australia's Eastern Standard Time. Users in Australia, of course, may prefer to do the converse. Since Coral8's time zone database is user-customizable, customers may choose the abbreviations that they wish, as long as each time zone's abbreviations do not overlap the abbreviations for any other time zone.

Coral8 Time Zone Database

Coral8 stores information about each time zone, including the abbreviation for the time zone (e.g. "PDT" for "Pacific Daylight Time"), the start date of Daylight Savings Time (if applicable) and the end date of Daylight Savings Time (if applicable), as a table of Comma-Separated Values in a file. We loosely refer to this table (and the file that contains it) as the "Coral8 time zone database". The file is named "c8_timezones.csv" and a copy is stored in the server plugins directory and the studio plugins directory.

There are 11 fields in the Coral8 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", e.g. 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 "Date Rule" is a specially formatted string used to indicate a day of the year for transitions (e.g. 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

This section describes the 11 fields in each row of the time zone database.

- ID - Contains the identifying label for time zone regions. 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 will suffice as long as it is unique within the time zone database. Each entry must have a region ID.
- STD Abbreviation - This contains the abbreviation of the time zone. The "STD" means "Standard Time zone Designator". This is the familiar "PST", "EST", etc. and is the standard time when daylight savings is not in effect. Abbreviations should consist only of upper case letters while the name may be any descriptive string. Lower case abbreviations, if present, will be capitalized. This entry is required.
- 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, will be capitalized. 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.
- 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 sames 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 will be 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

⚠ The Coral8 Time Zone Database stores only one set of information about each time zone. This means that when rules (e.g. for daylight savings time) are updated, Coral8 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 Coral8 time zone database, Coral8 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 Coral8 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 input or output.
- DST abbreviation maps to "TZD" as well. Use 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 as the user desires.

Standard and Daylight savings time abbreviations may be used interchangeably. The Coral8 software will, 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