HCL Informix 15.0.0

HCL Informix Change Data Capture API Programmer's Guide

Contents

Chapter 1. Change Data Capture API Programmer's Guide

The *HCL® Informix® Change Data Capture API Programmer's Guide* describes how to program applications to process changed data from HCL Informix® databases using the HCL Informix® Change Data Capture API.

This information is intended for application programmers.

Getting started with the Change Data Capture API

These topics describe the API and how to use it.

The Change Data Capture API

The API allows external client applications to capture transactional data from HCL Informix® databases.

The API provides functions to capture transactional data. You can use a variety of clients to run these functions, such as, JDBC, ODBC, ESQL/C, and DB-Access. The data is returned as CDC records by standard HCL® Informix® smart large object read functions. How the captured data is processed depends on your application. For example, you can write an application to replicate data from the HCL Informix® database to another, heterogeneous, database.

The following types of operations are captured:

- INSERT
- DELETE
- UPDATE
- TRUNCATE

The API starts capturing transactions from the current logical log and processes all transactions sequentially. The first time you start capturing data for a particular table, data capture starts at the current log position. If you later stop capture and the restart it, you can restart at the point in the logical logs where data capture was stopped. You cannot go backwards in time through the logical logs to capture the history of the table or perform random seeking in the logical logs.

At the beginning of data capture for a table, the API provides the table schema information that you can use in your application to create a target table. However, any changes to the table schema after data capture begins are not captured by the API.

The API can only provide data as that data is changing; it does not provide an initial snapshot of the contents of the table. If you need a populated target table, you can externally load the existing data to the target table. Alternatively, you can create dummy updates to the table for each row so that the API can capture those updates and populate the target table. Because logging is enabled, dummy updates produce logical log records.

The API does not capture changes to table schemas or any other database changes. If you attempt to alter a table while data capture is active, the alter process fails.

The API can capture data only from databases that have logging enabled.

Change Data Capture API components

The API consists of functions, a system database, CDC records, and error codes.

Functions

functions are built-in SQL functions that you run by using the EXECUTE FUNCTION statement. You use these functions to control data capture. The cdc_opensess() function returns the CDC session ID, which is a smart large object file descriptor that you use to retrieve captured data. The cdc_startcapture() function specifies the table from which to capture data. Other functions specify to start or end data capture.

You must call functions from a client application. You cannot call this function from a user-defined routine that runs within the database server.

System database

The **syscdc** system database contains the functions and system tables. The system tables store information about API error codes and record types.

Error codes

The API functions return error codes. Most of the functions return an error code both if they succeed or fail. The API error codes are listed in the **syscdcerrcodes** table of the **syscdc** database. You can query the **syscdcerrcodes** table to determine whether the function failed and if so, why it failed.

Smart large object read functions

You use smart large object read functions to read the captured data, by passing the smart large object file descriptor provided by the cdc_opensess() function. Smart large object read functions are not part of the API; you can use smart large object read functions such as mi_lo_read() or ifx_lo_read().

CDC Records

The (CDC) records are returned by smart large object read functions and provide information about the transaction currently being captured as well as the actual captured data.

Smart large object read functions

You use smart large object read functions to transfer captured data to a buffer where your application can access it.

You can use any of the smart large object read functions that are listed in the following table, depending on your application language. You must use the same smart large object read function for all read calls during a particular session. Using different functions in the same session can result in incomplete delivery of captured data.

Table 1. Smart large object read functions

Read function	Arguments	Informix® API	Application language
mi_lo_read()	A pointer to a connection descriptor		Use in a C language application.

Table 1. Smart large object read functions

(continued)

Read function	Arguments	Informix® API	Application language
	A smart large object file descriptor		
	A data buffer		
	The maximum number of bytes to read		
ifx_lo_read()	A smart large object file descriptor	ODBC	Use in an ODBC application.
	A data buffer		
ifx_lo_read()	A smart large object file descriptor	ESQL/C	Use in a C language application.
	A data buffer		
	The maximum number of bytes to read		
	A pointer to an error code		
IfxLoRead()	A smart large object file descriptor	JDBC	Use in a Java™ application.
	A data buffer		
	The maximum number of bytes to read		
IfxBlob.Read()	A data buffer	.NET	Use in a .NET application.

Read timeout

If no captured data is available to retrieve, the read call waits for data for the timeout period that is specified by the cdc_opensess() function. If the timeout period is exceeded, a CDC_REC_TIMEOUT record is returned to the read call. The read call passes the CDC_REC_TIMEOUT record into the data buffer and returns successfully.

Read buffer size

The size of the buffer that is specified in the read call must be at least 128 bytes. The maximum size of a read buffer is 2 GB. You can calculate the approximate minimum size of the buffer for your application by calculating the largest possible CDC record size, for example, a CDC_REC_INSERT record, and multiplying that value times the maximum number of records to return per read call that you specify in the cdc_opensess() function.

Amount and structure of data returned

The amount of data that is returned by a read call is limited by the size of the buffer that is specified in the read call and the maximum number of records to return. No more than the maximum number of records is returned by one read call, even if the number of bytes contained in those records is less than the maximum number of bytes allowed by the read call. However, no more than the maximum number of bytes allowed by the read call is returned, even if the number of records returned is less than the maximum number allowed.

The amount of data that is returned by smart large object read functions can differ from the size of the read buffer. The data that is returned is structured into CDC records. The number of CDC records that are returned by a smart large object read function varies and might not be an integral number. If a read call returns only part of a CDC record, the next read call returns more data for that record. The application must merge the parts of the record together. The Change Data Capture sample program provides examples of merging parts of records.

Smart large object file descriptor

The value for the smart large object file descriptor argument in the read functions is the CDC session ID returned by the cdc_opensess() function.

Smart large object read function for the Informix® .NET Provider

The smart large object read function for .NET works differently than for other client APIs. The following pseudo code illustrates the basic structure for reading smart large objects with .NET:

```
conn = new IfxConnection(..)// to SYSCDC database
execute function informix.cdc_opensess() // on the same connection
IfxBlob( IfxConnection connection )// construct it using the same connection
IfxBlob.Open(ReadOnly) // open it
IfxBlob.Read(long plofd, byte[] buff)
```

Related information

Smart large object examples on page

Smart large objects on page

IfxBlob class on page

Accessing a smart large object on page

Handling smart large objects on page 9

CDC record sequence numbers

Most (CDC) records returned to the client contain a sequence number.

The sequence number associated with a CDC record is a BIGINT data type.

The CDC record sequence number is not necessarily the same as the LSN of the HCL Informix® logical log that is being captured.

You can compare sequence numbers for CDC records that are returned for the same transaction. Within a transaction, the sequence numbers of CDC records returned increase over time. For most types of records, lower sequence number indicates that the CDC record was returned earlier than a CDC record with a higher sequence number. However, for a CDC_REC_DISCARD record type, the sequence number indicates from where to discard records.

You can compare the sequence numbers of CDC_REC_BEGINTX records or the sequence numbers of CDC_REC_COMMTX records for different transactions. Each committed transaction has one CDC_REC_BEGINTX record and one CDC_REC_COMMTX record. The sequence numbers for the CDC_REC_BEGINTX and CDC_REC_COMMTX records are in monotonic order. A lower sequence number indicates that the associated transaction was begun or committed earlier than a transaction associated with a higher sequence number.

Data for capture

You can capture most HCL Informix® data types. You can specify the data to capture at the column level.

The following data types are not supported for data capture:

- Simple large objects (TEXT and BYTE data types)
- User-defined data types
- Collection data types (SET, MULTISET, LIST, and ROW data types)

Specifying what data to capture

You specify a table and which columns from that table to capture with the cdc_startcapture() function. You must run the cdc_startcapture() function once for each table that you want to capture. For information about which tables and columns are currently being captured, look in the **syscdctabs** table.

Ending capture of a table

To stop data capture of a specific table, run the cdc_endcapture() function. After you run cdc_endcapture() function, information about that table is removed from the **syscdctabs** table.

Preparing to use the Change Data Capture API

Before you can start using the API, you must prepare the database and the database server.

About this task

Perform the following tasks to prepare for using the API:

1. Turn on logging for all databases from which you intend to capture data changes by running the cdc_set_fullrowlogging() function with the *logging* argument set to 1.
2. Run the following script as user **informix** from the `$INFORMIXDIR/etc` directory:
   Example
   syscdcv1.sql
3. Verify that the **syscdcv1** database exists by creating a connection to it, as user **informix**.

For example, you can use DB-Access to connect to the **syscdcv1** database.

4. Set the **DB_LOCALE** environment variable to be the same as the locale of the database from which you want to capture data.

Related information

Writing an application to capture data changes on page 8

Writing an application to capture data changes

Use the functions to control the data capture process. Process CDC records to extract the data. Query **syscdc** tables to retrieve the symbolic names and descriptions of CDC records and errors.

Before you begin

Complete the prerequisite tasks to prepare for using the API.

Your application should contain the following structures and functions:

- A structure to store table schema information. You use the table schema to parse the column data.
- A function to interpret the table schema information and populate the table schema structure. You can obtain the table schema information from the CDC_REC_TABSCHEMA record.
- A function to retrieve and parse the column values from the data buffer.
- A function to handle errors. You can query the **syscdcerrcodes** table to determine the symbolic name and description of the error code.

About this task

Include the following tasks in your application to capture data changes:

1. As user **informix**, connect to the **syscdcv1** database on the database server to which the client is currently connected.
2. Open a capture session by running the cdc_opensess() function.

   Result

   The cdc_opensess() function returns a session ID.
3. Enable full-row logging for each table from which you want to capture data by running the cdc_set_fullrowlogging() function.
4. Specify which data to capture by running the cdc_startcapture() function.

   Run this function for each table from which you want to capture data.
5. Start the capture process by running the cdc_activatesess() function.

   Result

   CDC records, including those that contain captured data, are returned to the application.
6. Read the CDC records containing captured data with a smart large object read function such as mi_lo_read() by passing the session ID as the large object file descriptor.

   Use the same smart large object read function for all read calls.

7. Parse the data by column values.
   If you are writing your application in Java™, you can use the IfxToJavaType class to convert a byte stream of the Informix® representation of a data type to the appropriate Java™ data type and value.
8. Stop capturing data by running the cdc_endcapture() function for each table.
9. Disable full-row logging by running the cdc_set_fullrowlogging() function for each table.
   Make sure that no other applications or processes are dependent on full-row logging before your disable it.
10. Close the capture session by running the cdc_closesess() function.

Related information

Preparing to use the Change Data Capture API on page 7

Change Data Capture sample program on page 39

Change Data Capture functions on page 11

Handling errors

To process errors that are returned by functions, reference error numbers by looking up their symbolic names in the **syscdcerrcodes** table.

About this task

Add code to your application to handle possible error conditions.

1. Declare error code variables for the types of errors that you intend to process separately.
2. Query the **syscdcerrcodes** table to find the error number corresponding to each of the symbolic names of the error codes.
3. Set the error code variables to the error numbers.
4. Add code to handle each error condition.

What to do next

You can use the cdc_errortext() function to return the error text for a specified symbolic name.

Related information

Change Data Capture error codes on page 34

Handling smart large objects

The Change Data Capture API does not directly support the retrieval of smart large object column data from a captured BLOB or CLOB row. You must use the or client API smart large object read functions to retrieve smart large objects.

To retrieve the data in a smart large object column, follow these general steps:

1. Retrieve the data row that contains the smart large object with the Change Data Capture API.
2. Extract the values of columns that uniquely identify the data row, such as the primary key or a unique constraint.
3. Run an SQL SELECT statement with the identifying values to retrieve the data row.
4. Open the smart large object from the column in the data row.
5. Retrieve the smart large object data by using one of the following types of smart large object functions:

   Choose from:

   - functions, such as mi_lo_read(), mi_lo_to_buffer(), or mi_lo_to_file()
   - SQL functions such as LOTOFILE()
   - ESQL/C functions such as ifx_lo_read()
6. Close the smart large object.

Related information

Smart large object examples on page

Smart large objects on page

IfxBlob class on page

Accessing a smart large object on page

Smart large object read functions on page 4

Restarting data capture

You can restart data capture where the last data capture session ended.

Before you begin

The restart position is the sequence number of a CDC record that was returned in the previous data capture session. You can use the sequence number of the last CDC record processed in the previous data capture session. However, to preserve transactional integrity, you should determine last transaction for which a commit or rollback operation was not processed and restart capture at the beginning of that transaction. In this case, the restart position is the lowest sequence number of the CDC_REC_BEGINTX records for incomplete transactions. To avoid reprocessing already committed transactions, you should also determine the largest sequence number of the CDC_REC_COMMTX record that you have already processed in a previous data capture session.

About this task

To restart data capture:

1. Determine the restart position.

   To preserve transactional integrity:

   a. Find all captured transactions that did not return a CDC_REC_COMMTX or CDC_REC_RBTX record.
   b. Compare the sequence numbers of the CDC_REC_BEGINTX records for the incomplete transactions. The lowest sequence number is the restart position.
2. Open a new capture session by running the cdc_opensess() function.
3. Run the cdc_startcapture() function for the table on which you want to restart capturing data.

4. Active the session by running the cdc_activatesess() function. Pass the appropriate sequence number as the *position* argument.
   Result
   Data capture restarts for the table at the last transaction that was processed.
5. Discard any transactions whose CDC_REC_COMMTX sequence number is less than that of the CDC_REC_COMMTX record with the largest sequence number that you processed in the previous data capture session.

Related information

The cdc_activatesess() function on page 11

Monitoring data capture

You can monitor the status of data capture by running the onstat -g cdc command.

About this task

To view the current status of a data capture session, run the onstat -g cdc command. For this command, and all other onstat -g cdc command options, you can specify a single session or view information about all current sessions.

To view the status of session buffers, run the onstat -g cdc bufm command.

To view information about session configuration, run the onstat -g cdc config command.

To view information about tables currently being captured, run the onstat -g cdc table command. You can provide a single table name or view information for all tables.

Related information

onstat -g cdc on page 36

Change Data Capture functions

These topics describe the functions.

Related information

Writing an application to capture data changes on page 8

The cdc_activatesess() function

For an open capture session, starts capturing data from the specified log and log position.

The **syscdcsess** table is updated when the session is activated.

Syntax

```
cdc_activatesess(session_ID ,position)
```

Function arguments

Table 2. The cdc_activatesess() arguments

Argument	Data Type	Description
session_ID	INTEGER	The session ID of the open capture session for which to start capturing data.
position	BIGINT	Must be `0` or the restart position.

Usage

After you open a session with the cdc_opensess() function, you use the cdc_activatesess() function to start capturing data at the specified log position. If you are starting data capture on a table for the first time, the *position* must be `0`. If you have previously performed data capture, you can restart data capture where it left off by specifying a sequence number of a CDC record returned in the previous capture session.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Return values

If successful, returns `0`.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

Related information

Restarting data capture on page 10

The cdc_closesess() function

Closes a capture session that is associated with the specified session ID.

Any resources used by the capture session are released. The rows in the **syscdctabs** and **syscdcsess** tables containing the specified session ID are deleted.

Syntax

```
cdc_closesess(session_ID )
```

Function argument

Table 3. The cdc_closesess() argument

Argument	Data Type	Description
session_ID	INTEGER	The session ID of the capture session that you want to close.

Usage

Use the cdc_closesess() function to close a capture session that you no longer need. If the capture session was active, all data capture is immediately stopped when the session is closed.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Return values

If successful, returns `0`.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

The cdc_deactivatesess() function

Stops capturing data for an active capture session.

The **syscdcsess** table is updated to show that the capture session is not active.

Syntax

```
cdc_deactivatesess(session_ID )
```

Function argument

Table 4. The cdc_deactivatesess() argument

Argument	Data Type	Description
session_ID	INTEGER	The session ID of the capture session that you want to deactivate.

Usage

Use the cdc_deactivatesess() function to stop capturing data for a specific capture session. However, you do not need to run the cdc_deactivatesess() function if you run the cdc_closesess() and the cdc_deactivesess() functions.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Return values

If successful, returns `0`.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

The cdc_endcapture() function

Ends capture for a specified table.

The row in the **syscdctabs** table associated with the specified session ID and table is deleted.

Purpose

```
cdc_endcapture( session_ID,MBZ, "database: owner.table_name" )
```

Function arguments

Table 5. The cdc_endcapture() arguments

Argument	Data Type	Description
session_ID	INTEGER	The session ID of an open capture session.
MBZ	BIGINT	Must be `0`. Reserved.
database:*owner*.*table_name*	LVARCHAR	The qualified name of the table from which to capture data. The qualified name includes the following elements: ***database*** The name of the database in which the table exists. ***owner*** The name of the owner of the table. ***table*** The name of the table

Usage

Use the cdc_endcapture() function to stop capturing data from a specific table. This function does not affect the session status; the session remains open and active.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Return values

If successful, returns `0`.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

The cdc_errortext() function

Returns the error message text corresponding to the specified symbolic error name.

Symbolic error names are listed in the **syscdcerrcodes** table in the **syscdc** database.

Syntax

```
cdc_errortext('error_name ','locale_name ')
```

Function arguments

Table 6. The cdc_errortext() arguments

Argument	Data Type	Description
error_name	LVARCHAR	The symbolic name of the error.
locale_name	LVARCHAR	The name of the locale in which to display the error text. If locale name parameter is SQL NULL or a string of `0` length ("") the default locale is used.

Usage

Use the cdc_errortext() function to return the error text for an error that you received from another CDC function. Not all error texts are available in all locales. If the cdc_errortext() function does not return the text in the locale you specified, try to run the function again with a different locale, such as `'en_us.819'` or `'en_us.0333'`.

Return values

If successful, returns SQLCODE `0` and the error message text.

If unsuccessful, returns with a nonzero SQLCODE:

- 23109: Invalid locale specification.

  The locale name is not correct or the specified locale was not found.
- 1824: Message cannot be found.

The locale is valid but the message was not found in the message file for that locale. Specify a different locale, such as **en_us.0333**.

- Other SQLCODES represent internal errors.

Example

Example

The following example returns the error text for the error CDC_E_TABCAPTURED in the **en_us.0333** locale:

```
> select cdc_errortext('CDC_E_TABCAPTURED', 'en_us.0333') from syscdcvers;


(expression)  The specified table is already being captured by the CDC session.


1 row(s) retrieved.
```

Related information

Change Data Capture error codes on page 34

The cdc_opensess() function

Opens a capture session and creates a session ID.

A row is inserted into the **syscdcsess** table for the session.

Syntax

cdc_opensess("*server_name* ",*session_ID*,*timeout* , *max_recs*,*major_version* ,*minor_version*)

Function arguments

Table 7. The cdc_opensess() arguments

Argument	Data Type	Description
server_name	LVARCHAR	The name of the server. Must be the server to which the client application that is calling the cdc_opensess() function is connected.
session_ID	INTEGER	Must be `0`.
timeout	INTEGER	Specifies the timeout behavior of a read call on the captured data:

Table 7. The cdc_opensess() arguments (continued)

Argument	Data Type	Description
		<0 Do not timeout. **0** Return immediately if no data is available. **1 or more** The number of seconds to wait for data before timing out.
max_recs	INTEGER	The maximum number of CDC records to return per read function call. This value takes precedence over the maximum number of bytes to return that is specified in the smart large object read function.
major_version	INTEGER	The major version number of the API. Must be `1`.
minor_version	INTEGER	The minor version number of the API. Must be `1` for new applications. Can be `0` for existing applications.

Usage

Use the cdc_opensess() function to open a communication session between the client application and the database server. The session ID returned by the cdc_opensess() is the smart large object file descriptor that you supply to the smart large object read function. To start capturing data, you must then use the cdc_activatesess() function and the cdc_startcapture() function.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Important: If you have multiple applications that use the API and connect to the same Informix® server, all applications must use the same values for the *major_version* and *minor_version* arguments.

Return values

If successful, returns an integer that is the session ID.

If unsuccessful, returns an integer corresponding to an error code.

The cdc_recboundary() function

Restarts data capture from the beginning of the CDC record currently being returned.

Syntax

```
cdc_recboundary(session_ID )
```

Function argument

Table 8. The cdc_recboundary() argument

Argument	Data Type	Description
session_ID	INTEGER	The session ID of the open capture session.

Usage

Use the cdc_recboundary() function if you need to restart capture from the beginning of the current log record.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Return values

If successful, returns a positive integer representing the number of complete or partial log records that were captured but skipped during the current session.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

The cdc_set_fullrowlogging() function

Enables or disables full-row logging for a table.

Purpose

You must run this function to enable full-row logging on a table before you can start capturing data from it.

The **DB_LOCALE** environment variable must be set to the same locale as the database locale when you run this function.

```
cdc_set_fullrowlogging ( " database : owner . table_name " , logging )
```

Function arguments

Table 9. The cdc_set_fullrowlogging() arguments

Argument	Data Type	Description
database:*owner*.*table_name*	LVARCHAR	The qualified name of the table. The qualified name includes the following elements:

Table 9. The cdc_set_fullrowlogging() arguments (continued)

Argument	Data Type	Description
		database The name of the database in which the table exists. ***owner*** The name of the owner of the table. ***table*** The name of the table.
logging	INTEGER	• `0` Disable full-row logging • `1` Enable full-row logging

Usage

Use the cdc_set_fullrowlogging() function to enable full-row logging on a table from which you intend to perform data capture. This function must be run as user **informix**. You must stop capturing data from a table using cdc_endcapture() before you disable full-row logging using cdc_set_fullrowlogging(). Without cdc_endcapture(), the disabling of full-row logging will fail with error *"19816: Cannot perform this operation on a table defined for replication"*. Placeholder updates are fully logged. Logging remains enabled when the session is closed.

You can check the status of logging by running the oncheck -pT command. The Tblspace flags section of the oncheck -pT output shows the following text if logging is enabled: `TBLspace flagged for Log Snooping`.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Note: When *cdc_set_fullrowlogging()* is invoked, the specified table is set to FULL ROW LOGGING mode. Now that we support CDC applications on secondary servers as well, it is recommended to set the database to FULL ROW LOGGING mode only once at the beginning by the CDC application to avoid accidental unsetting by other CDC application.

Return values

If successful, returns `0`.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

The cdc_startcapture() function

Specifies the data to start capturing from a table.

If the capture session is both open and active (you have run the cdc_activatesess() function), data capture starts immediately on the specified columns in the specified table. Otherwise, data capture starts when you activate the open capture session.

The **DB_LOCALE** environment variable must be set to the same locale as the database locale when you run this function.

A row is added in the **syscdctabs** table associated with the specified session ID and table.

Syntax

```
cdc_startcapture(session_ID ,MBZ, "database: owner.table_name" ," column_name " ,user_data)
```

Function arguments

Table 10. cdc_startcapture() arguments

Argument	Data Type	Description
session_ID	INTEGER	The session ID of an open capture session.
MBZ	BIGNIT	Must be `0`. Reserved.
database:*owner*.*table_name*	LVARCHAR	The qualified name of the table from which to capture data. The qualified name includes the following elements: ***database*** The name of the database in which the table exists. ***owner*** The name of the owner of the table. ***table*** The name of the table.
column_name	LVARCHAR	A comma-separated list of column names in the specified table, from which to capture data.
user_data	INTEGER	The table identifier.

Usage

Use the cdc_startcapture() function to specify a table and columns within that table from which to start capturing data. You cannot include columns with simple large objects, user-defined data types, or collection data types.

The table identifier is a number you use in your application to uniquely identify each table that will participate in data capture.

You must call this function from a client application. You cannot call this function from a user-defined routine that runs within the database server.

Return values

If successful, returns `0`.

If unsuccessful, returns an integer corresponding to an error code and updates the **syscdcsess** table with the error information.

Change Data Capture records

The Change Data Capture (CDC) records are returned by smart large object read functions and contain the captured data and information about the transaction currently being captured.

Your application cannot depend on a specific number of records being returned by smart large object read calls. The number of CDC records that are returned in a read call is not predictable and might not be an integral number. If a read call returns only part of a CDC record, the next read call returns more data for that record. Your application must merge the parts of the record that are returned in multiple read calls. The Change Data Capture sample program provides examples of merging parts of records.

Format of CDC records

The (CDC) records contain a header that is common to all records, followed by a specific header for the type of CDC record.

The CDC_REC_INSERT, CDC_REC_DELETE, CDC_REC_UPDBEF, and CDC_REC_UPDAFT records also contain column data.

The header common to all CDC records describes the size and type of the CDC record.

Table 11. The header common to all CDC records

Section	Size	Description
Header size	4 bytes	The number of bytes in the common and CDC record-specific headers.
Payload size	4 bytes	The number of bytes of data in the record after the common and CDC record-specific headers.
Packet scheme	4 bytes	The packetization scheme number of one of the packetization schemes contained in the **syscdcpacketschemes** table. The only packetization scheme is `66`, CDC_PKTSCHEME_LRECBINARY.
Record number	4 bytes	The record number of one of the CDC records contained in the **syscdcrectypes** table.

Related information

The CDC_REC_BEGINTX record on page 22

The CDC_REC_COMMTX record on page 23

The CDC_REC_DELETE record on page 23

The CDC_REC_DISCARD record on page 24

The CDC_REC_ERROR record on page 26

The CDC_REC_INSERT record on page 27

The CDC_REC_RBTX record on page 28

The CDC_REC_TABSCHEMA record on page 29

The CDC_REC_TIMEOUT record on page 29

The CDC_REC_TRUNCATE record on page 30

The CDC_REC_UPDAFT record on page 30

The CDC_REC_UPDBEF record on page 32

The CDC_REC_BEGINTX record

Indicates the beginning of a transaction.

The header for the CDC_REC_BEGINTX record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 12. Format of the CDC_REC_BEGINTX record

Section	Size	Description
Sequence number	8 bytes	The sequence number of the record.
Transaction ID	4 bytes	The transaction ID.
Start time	8 bytes	The UTC time at which the transaction began, in time_t format.
User ID	4 bytes	The operating system user ID of the user who started the transaction.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_COMMTX record

Indicates that a transaction has been committed.

The header for the CDC_REC_COMMTX record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 13. Format of the CDC_REC_COMMTX record

Section	Size	Description
Sequence number	8 bytes	The sequence number of the record.
Transaction ID	4 bytes	The transaction ID.
Commit time	8 bytes	The UTC time at which the transaction was committed, in time_t format.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_DELETE record

Provides the row that was removed as a result of a DELETE operation.

The CDC_REC_DELETE record consists of these fields:

- The common header.
- The record-specific header.
- Fields listing the size of each variable-length column in the row, if any.
- Column data for each fixed-length column, if any.
- Column data for each variable-length column, if any.

The value in the header size field in the common header represents the number of bytes occupied by the combination of the common header, the record-specific header, and the fields listing the size of variable-length columns.

The value in the payload size field in the common header represents the number of bytes of the column data for both fixed-length and variable length columns.

The record-specific header

The header specific to the CDC_REC_DELETE record follows the common header.

Table 14. The CDC_REC_DELETE record header

Section	Size	Description
Sequence number	8 bytes	The sequence number associated with the DELETE operation.
Transaction ID	4 bytes	The transaction ID.
User data	4 bytes	The table identifier passed to the cdc_startcapture() function and stored in the **syscdtabs** table.
Flags	4 bytes	Reserved.

Variable-length column size fields

If there are variable-length columns in the row being deleted, a 4-byte field for each of those columns appears containing the column size. The order of the column size fields is the same as the order of the columns in the CDC_REC_TABSCHEMA record.

Fixed-length column data

The data from the fixed-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Variable-length column data

The data from the variable-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_DISCARD record

Indicates that some operations of the transaction should be discarded.

CDC records for the same transaction that follow this record should be discarded.

The header specific to the CDC_REC_DISCARD record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 15. Format of the CDC_REC_DISCARD record

Section	Size	Description
Sequence number	8 bytes	The sequence number of the record. Any CDC records that have the same transaction ID value and that have a sequence number greater than or equal to this sequence number should be discarded.
Transaction ID	4 bytes	The transaction ID.

Example

Example

The following example creates a savepoint, inserts data, rolls back the transaction to the savepoint, then inserts and commits data:

```
> begin work;
Started transaction.
> savepoint sp1;
Savepoint set.
> insert into t1(c1) values (2000);
1 row(s) inserted.
> insert into t1(c1) values (2001);
1 row(s) inserted.
> rollback to savepoint sp1;
Transaction rolled back to savepoint.
> insert into t1(c1) values(2);
1 row(s) inserted.> commit;
Data committed.
```

The following records are generated:

```
Got Record type   CDC_REC_BEGINTX. Size =    0 LSN = 28:0xaa5018. TXID = 25
    Time = 2016-03-30 11:59:26
Raw Data:  (0/0x0 bytes at address 0xc54420)
bytesread is 58 loreaderr is 0 SQLCODE 0
Got Record type    CDC_REC_INSERT. Size =   22 LSN = 28:0xaa5050. TXID = 25
    TabID = 0
Raw IUD Data:  (22/0x16 bytes at address 0xc54434)
00 00 07 d0 00 20 20 20 20 20 20 20 20 20 80 00 .....          ..
00 00 80 00 00 00                               ......
    Column Value = 2000
    Column Value = 'NULL'
    Column Value = NULL
    Column Value = NULL
bytesread is 58 loreaderr is 0 SQLCODE 0
Got Record type    CDC_REC_INSERT. Size =   22 LSN = 28:0xaa50a8. TXID = 25
    TabID = 0
Raw IUD Data:  (22/0x16 bytes at address 0xc54434)
00 00 07 d1 00 20 20 20 20 20 20 20 20 20 80 00 .....          ..
00 00 80 00 00 00                               ......
```

```
    Column Value = 2001
    Column Value = 'NULL'
    Column Value = NULL
    Column Value = NULL
bytesread is 28 loreaderr is 0 SQLCODE 0
Got Record type   CDC_REC_DISCARD. Size =    0     Total record size = 0
    LSN = 28:0xaa5050
    TXID = 25
bytesread is 58 loreaderr is 0 SQLCODE 0
Got Record type    CDC_REC_INSERT. Size =   22 LSN = 28:0xaa5150. TXID = 25
    TabID = 0
Raw IUD Data:  (22/0x16 bytes at address 0xc54434)
00 00 00 02 00 20 20 20 20 20 20 20 20 20 80 00 .....         ..
00 00 80 00 00 00                               ......
    Column Value = 2
    Column Value = 'NULL'
    Column Value = NULL
    Column Value = NULL
bytesread is 36 loreaderr is 0 SQLCODE 0
Got Record type    CDC_REC_COMMTX. Size =    0 LSN = 28:0xaa51a8. TXID = 25
    Time = 2016-03-30 11:59:47
Raw Data:  (0/0x0 bytes at address 0xc54420)
```

The CDC_REC_DISCARD record indicates that for transaction 25, the records starting from LSN 28:0xaa5050 and until the CDC_REC_DISCARD record are discarded. The insert of the values "2000" and "2001" are discarded because the client rolled back to the savepoint. The records after the CDC_REC_DISCARD record are valid actions in the transaction. For example, the CDC_REC_INSERT at 28:0xaa5150, which corresponds to the insert of the value "2", is committed.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_ERROR record

Indicates that an error occurred and the session is no longer valid.

The header specific to the CDC_REC_ERROR record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 16. Format of the CDC_REC_ERROR record

Section	Size	Description
Flags	4 bytes	Hexadecimal flag: • `0x1` indicates that the capture session is no longer valid and the only valid operation is to run the cdc_closesess() function to close the session. • any other value indicates that the session is still valid.

Table 16. Format of the CDC_REC_ERROR record (continued)

Section	Size	Description
Error code	4 bytes	The error code.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_INSERT record

Provides the row that resulted from an INSERT operation.

The CDC_REC_INSERT record consists of these fields:

- The common header.
- The record-specific header.
- Fields listing the size of each variable-length column in the row, if any.
- Column data for each fixed-length column, if any.
- Column data for each variable-length column, if any.

The value in the header size field in the common header represents the number of bytes occupied by the combination of the common header, the record-specific header, and the fields listing the size of variable-length columns.

The value in the payload size field in the common header represents the number of bytes of the column data for both fixed-length and variable length columns.

The record-specific header

The header specific to the CDC_REC_INSERT record follows the common header.

Table 17. The CDC_REC_INSERT record header

Section	Size	Description
Sequence number	8 bytes	The sequence number associated with the INSERT operation.
Transaction ID	4 bytes	The transaction ID.
User data	4 bytes	The table identifier passed to the cdc_startcapture() function and stored in the **syscdtabs** table.

Table 17. The CDC_REC_INSERT record header (continued)

Section	Size	Description
Flags	4 bytes	Reserved.

Variable-length column size fields

If there are variable-length columns in the row being inserted, a 4–byte field for each of those columns appears containing the column size. The order of the column size fields is the same as the order of the columns in the CDC_REC_TABSCHEMA record.

Fixed-length column data

The data from the fixed-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Variable-length column data

The data from the variable-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_RBTX record

Indicates that the transaction has been rolled back.

The header specific to the CDC_REC_RBTX record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 18. Format of the CDC_REC_RBTX record

Section	Size	Description
Sequence number	8 bytes	The sequence number associated with the ROLLBACK operation.
Transaction ID	4 bytes	The transaction ID.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_TABSCHEMA record

Describes the table from which data is being captured.

The value in the payload size field in the common header represents the number of bytes occupied by the column name and data type list.

The header specific to the CDC_REC_TABSCHEMA record follows the common header.

Table 19. Format of the CDC_REC_TABSCHEMA record

Section	Size	Description
User data	4 bytes	The table identifier that was specified in the cdc_startcapture() function for the table being captured.
Flags	4 bytes	Must be `0`.
Fixed-length size	4 bytes	The number of bytes of data in fixed-length columns in the table.
Fixed-length columns	4 bytes	The number of fixed-length columns in the table being captured. A `0` indicates that there are no fixed-length columns.
Variable-length columns	4 bytes	The number of variable-length columns in the table being captured. A `0` indicates that there are no variable-length columns.
Column names and data types	variable byte length	A comma-separated list of column names and data types in UTF-8 format. The column list conforms to the syntax of the column list in a CREATE TABLE statement. Names of any fixed-length columns appear before names of any variable-length columns. The number of columns equals the number of fixed-length columns plus the number of variable-length columns.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_TIMEOUT record

Indicates that the read call did not return data before the timeout period specified in the cdc_opensess() function.

The header specific to the CDC_REC_TIMEOUT record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 20. Format of the CDC_REC_TIMEOUT record

Section	Size	Description
Sequence number	8 bytes	The sequence number of the last data retrieved from the source database.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_TRUNCATE record

Indicates that a TRUNCATE operation was performed on a table.

The header specific to the CDC_REC_TRUNCATE record follows the common header. No data follows the headers; the payload size in the common header is `0`.

Table 21. Format of the CDC_REC_TRUNCATE record

Section	Size	Description
Sequence number	8 bytes	The sequence number associated with the TRUNCATE operation.
Transaction ID	4 bytes	The transaction ID.
User data	4 bytes	The table identifier passed to the cdc_startcapture() function and stored in the **syscdtabs** table.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_UPDAFT record

Provides the image of a row after an UPDATE operation.

The CDC_REC_UPDAFT record consists of these fields:

- The common header.
- The record-specific header.
- Fields listing the size of each variable-length column in the row, if any.

- Column data for each fixed-length column, if any.
- Column data for each variable-length column, if any.

The value in the header size field in the common header represents the number of bytes occupied by the combination of the common header, the record-specific header, and the fields listing the size of variable-length columns.

The value in the payload size field in the common header represents the number of bytes of the column data for both fixed-length and variable length columns.

The record-specific header

The header specific to the CDC_REC_UPDAFT record follows the common header.

Table 22. The CDC_REC_UPDAFT record header

Section	Size	Description
Sequence number	8 bytes	The sequence number associated with the UPDATE operation.
Transaction ID	4 bytes	The transaction ID.
User data	4 bytes	The table identifier passed to the cdc_startcapture() function and stored in the **syscdtabs** table.
Flags	4 bytes	Reserved.

Variable-length column size fields

If there are variable-length columns in the row being updated, a 4-byte field for each of those columns appears containing the column size. The order of the column size fields is the same as the order of the columns in the CDC_REC_TABSCHEMA record.

Fixed-length column data

The data from the fixed-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Variable-length column data

The data from the variable-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The CDC_REC_UPDBEF record

Provides the image of a row before an UPDATE operation.

The CDC_REC_UPDBEF record consists of these fields:

- The common header.
- The record-specific header.
- Fields listing the size of each variable-length column in the row, if any.
- Column data for each fixed-length column, if any.
- Column data for each variable-length column, if any.

The value in the header size field in the common header represents the number of bytes occupied by the combination of the common header, the record-specific header, and the fields listing the size of variable-length columns.

The value in the payload size field in the common header represents the number of bytes of the column data for both fixed-length and variable length columns.

The record-specific header

The header specific to the CDC_REC_UPDBEF record follows the common header.

Table 23. The CDC_REC_UPDBEF record header

Section	Size	Description
Sequence number	8 bytes	The sequence number associated with the UPDATE operation.
Transaction ID	4 bytes	The transaction ID.
User data	4 bytes	The table identifier passed to the cdc_startcapture() function and stored in the **syscdtabs** table.
Flags	4 bytes	Reserved.

Variable-length column size fields

If there are variable-length columns in the row being updated, a 4-byte field for each of those columns appears containing the column size. The order of the column size fields is the same as the order of the columns in the CDC_REC_TABSCHEMA record.

Fixed-length column data

The data from the fixed-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Variable-length column data

The data from the variable-length columns, if any, appears in the order that the corresponding columns are listed in the CDC_REC_TABSCHEMA record.

Related information

Format of CDC records on page 21

The syscdcrectypes table on page 33

The syscdc system database

The **syscdc** system database contains tables that store information about the API.

The **syscdc** database can only be accessed or connected to by the user **informix**. It uses the UTF-8 locale. You cannot alter the tables in the **syscdc** database; you can only query them.

The syscdcerrcodes table

Contains the error codes used by the API.

Use this table to look up the symbolic name and description that correspond to an error code.

Table 24. The syscdcerrcodes table

Column	Data Type	Description
errcode	INTEGER	Numeric value of the error.
errname	VARCHAR(16)	Symbolic name of the error.
errdesc	VARCHAR(127)	Error description.

Related information

Change Data Capture error codes on page 34

The syscdcrectypes table

Contains the record types used by the API.

Use this table to look up the symbolic name and description that correspond to a record code.

Table 25. The syscdcrectypes table

Column	Data Type	Description
recnum	INTEGER	Numeric value of the record type.
recname	VARCHAR(16)	Symbolic name of the record type.
recdesc	VARCHAR(127)	Record type description.

Related information

The CDC_REC_BEGINTX record on page 22

The CDC_REC_COMMTX record on page 23

The CDC_REC_DELETE record on page 23

The CDC_REC_DISCARD record on page 24

The CDC_REC_ERROR record on page 26

The CDC_REC_INSERT record on page 27

The CDC_REC_RBTX record on page 28

The CDC_REC_TABSCHEMA record on page 29

The CDC_REC_TIMEOUT record on page 29

The CDC_REC_TRUNCATE record on page 30

The CDC_REC_UPDAFT record on page 30

The CDC_REC_UPDBEF record on page 32

Change Data Capture error codes

If a function encounters a problem, it returns an error code. Most functions return `0` if they succeed.

Error numbers are not guaranteed to remain the same in subsequent releases. Always use the symbolic names in your application code. You can view the error message text corresponding to a symbolic error name by using the cdc_errortext() function.

Table 26. error codes

Symbolic Name	Description
CDC_E_OK	Operation succeeded.
CDC_E_NOCDCDB	The **syscdc** database does not exist.
CDC_E_APIVERS	The requested CDC API behavior version is not valid or is unsupported.

Table 26. error codes (continued)

Symbolic Name	Description
CDC_E_NODB	The specified database does not exist.
CDC_E_DBNOTLOGGED	The specified database is not logged.
CDC_E_NOTAB	The specified table does not exist.
CDC_E_TABPROPERTIES	The table properties do not support capture: it is a temporary table, a view, or otherwise not logged.
CDC_E_NOCOL	The specified column does not exist.
CDC_E_NOSES	The specified CDC session does not exist.
CDC_E_NOREOPEN	The CDC session cannot be reopened.
CDC_E_TABCAPTURED	The specified table is already being captured by the CDC session.
CDC_E_TABNOTCAPTURED	The specified table is not being captured by the CDC session.
CDC_E_ARGNULL	An argument to the function has the SQL NULL value, which is not allowed.
CDC_E_LSN	Data at the requested log sequence number is unavailable for capture.
CDC_E_DUPLSESS	A CDC session is already active.
CDC_E_ARG	A parameter passed to the function is not valid.
CDC_E_ARG1	The first parameter passed to the function is not valid.
CDC_E_ARG2	The second parameter passed to the function is not valid.
CDC_E_ARG3	The third parameter passed to the function is not valid.
CDC_E_ARG4	The fourth parameter passed to the function is not valid.
CDC_E_ARG5	The fifth parameter passed to the function is not valid.
CDC_E_ARG6	The sixth parameter passed to the function is not valid.
CDC_E_INTERNAL	Internal error. Contact IBM® Support.
CDC_E_NOMEM	Memory allocation failed.
CDC_E_MUSTCLOSE	The CDC capture session cannot continue and must be closed.
CDC_E_BADSTATE	The resource state does not allow the attempted operation.
CDC_E_BADCHAR	A byte sequence that is not a valid character in the character code set was encountered.
CDC_E_INTERRUPT	The CDC session was interrupted.
CDC_E_UNIMPL	Unimplemented feature.

Table 26. error codes (continued)

Symbolic Name	Description
CDC_E_LOCALEMISMATCH	The locale setting in the environment does not match the locale of the database.

Related information

The syscdcerrcodes table on page 33

The cdc_errortext() function on page 15

Handling errors on page 9

onstat -g cdc

Monitors the sessions involved in change data capture.

```
onstat -g cdc [{0 |sessionID}]{[{bufm|table[database:owner .table]}] [long]  | config}
```

Table 27. The onstat -g cdc syntax elements

Element	Purpose
bufm	Displays information about the buffers being used by the session, including: • The highest number of buffers used by the session. • The number of buffers currently being used by the session. • With the **long** option, the address of each allocated buffer.
config	Displays information about the session configuration, including: • The read timeout setting for the session, in seconds. • The maximum number of records returned by a read call.
database:*owner*.*table*	The fully-qualified name of the table for which to display information. The qualified name includes the following elements: • *database*: The name of the database in which the table exists. • *owner*: The name of the owner of the table. • *table*: the name of the table.
long	Provides additional detail for sessions, the **bufm** option, or the **table** option.
sessionID	Displays information for the specified session ID:

Table 27. The onstat -g cdc syntax elements (continued)

Element	Purpose
	• The associated SQL session ID. • The number of tables being captured by the session. • With the **long** option, information about the number of records processed by the session. If you do not specify a session ID, or if you specify a session ID of `0`, information for all sessions is displayed.
table	Displays information about the tables being captured, including: • The number of tables being captured in a session. • The full name of each table being captured. • The time when data capture on each table started. • With the **long** option, information about the captured columns for each table. If you specify a fully-qualified table name, only the information for that table is displayed. If you do not specify a table name, information for all tables is displayed.

Example

Examples

The following examples display sample output of the onstat -g cdc command with some of its options.

Example 1: Detailed session information

The following command generates output that shows detailed information about the session 159383591:

```
onstat -g cdc 159383591 long

CDC subsystem structure at 0x44252318
    CDC session structure at 0x4d8e0d00
        CDC session id: 159383591 (0x9800027)
        Associated SQL session id: 304
        Number of tables captured: 1
        State: ACTIVATED (0x50534555)
        Create time: 1238530254 (Tue Mar 31 15:10:54 2009)
        Open time: 1238530254 (Tue Mar 31 15:10:54 2009)
        Activate time: 1238530256 (Tue Mar 31 15:10:56 2009)
        Activate Sequence Number: 0x0
        Total client read calls: 9
        Last client read time: 1238530321 (Tue Mar 31 15:12:01 2009)
        Last Sequence Number returned to client: 0x150004b774
        Total number records examined: 4385
        Total number records kept (approximate): 1937
        Total number I/U/D records examined: 1046
        Total number I/U/D records kept (approximate): 582
        Client required to close: NO
        Read exit error code: 0
```

Example 2: Configuration information

The following command generates output that shows information about the configuration of open sessions:

```
onstat -g cdc config

CDC subsystem structure at 0x44252318
    CDC session structure at 0x4dba3d00
        CDC session id: 160432167 (0x9900027)
        Read Timeout (seconds): 3
        Maximum buffers per read call: 4
        Survive DATALOST errors: NO

    CDC session structure at 0x4d8e0d00
        CDC session id: 159383591 (0x9800027)
        Read Timeout (seconds): 3
        Maximum buffers per read call: 4
        Survive DATALOST errors: NO

    CDC session structure at 0x4c022d00
        CDC session id: 158335015 (0x9700027)
        Read Timeout (seconds): 3
        Maximum buffers per read call: 4
        Survive DATALOST errors: NO
```

Example 3: Buffer information

The following command generates output that shows information about the buffers being used by currently open sessions:

```
onstat -g cdc 0 bufm

CDC subsystem structure at 0x44252318
    CDC session structure at 0x4dba3d00
        CDC session id: 160432167 (0x9900027)

        Buffer Manager  at 0x4dba5028
            Number of allocated buffers high watermark: 268
            Number of currently allocated buffers: 267
            Minimum prepend for alloced buffers: 172

    CDC session structure at 0x4d8e0d00
        CDC session id: 159383591 (0x9800027)

        Buffer Manager  at 0x4d8e2028
            Number of allocated buffers high watermark: 271
            Number of currently allocated buffers: 270
            Minimum prepend for alloced buffers: 172

    CDC session structure at 0x4c022d00
        CDC session id: 158335015 (0x9700027)

        Buffer Manager  at 0x4c6e5028
            Number of allocated buffers high watermark: 269
            Number of currently allocated buffers: 267
            Minimum prepend for alloced buffers: 172
```

Example 4: Table information

The following command generates output that shows information about the session 158335015 for the table named **account**:

```
onstat -g cdc 158335015 table bank:pinch.account

CDC subsystem structure at 0x44252318
    CDC session structure at 0x4c022d00
        CDC session id: 158335015 (0x9700027)
        Captured Table Manager found at 0x4c048b20
        Number of tables captured: 1

            Captured Table structure at 0x4c6e5160
                Full Table Name: bank:pinch.account
                Version Sequence Number: 0xe00238388
                Time capture started: 1238530249 (Tue Mar 31 15:10:49 2009)
```

Related information

Monitoring data capture on page 11

Change Data Capture sample program

The Change Data Capture sample program provides an example of using the API to capture and process data.

Example

The sample program, `cdcapi.ec`, is located in the `INFORMIXDIR/demo/cdc` directory. The program creates an application that captures data from multiple tables. The program runs functions, reads CDC records, and displays the column values of the captured data to stdout. The program also queries the **syscdc** system tables to display information about CDC records and error messages. The program terminates when it encounters an error or a CDC_REC_TIMEOUT record.

The program has a command-line interface that you use to enter the database name, the table name, column names, and the timeout value.

This program requires that the getopt parser function is implemented on your computer.

Related information

Writing an application to capture data changes on page 8

Informix Change Streams API for Java

This topic demonstrates how to use the Informix Change Streams API to create data streams and capture changing data from the server using the Java programming language.

The Informix Change Streams client API allows you to easily stream changes made from a logged database in Informix into your Java application. The Change Streams client provides a high-level API that abstracts the details of the underlying streaming system. This allows you to get triggered events when there is a change to a specific table you instruct the API to watch. The API will trigger events in your application for changes to a database table. For more information, see Change Data Capture records for a description of the types of events the client can receive.

Informix Change Streams API currently allows you to configure the underlying Informix Change Data Capture feature (see Change data capture on page) and receive events from this system in well-defined objects. You can subscribe to events from one or more tables, specifying which columns of data you are interested in receiving.

The client API is built into a Java library (JAR file) with a dependency on the Informix JDBC 4.50.JC2 or newer database driver. Both libraries are packaged as part of the JDBC installation and the client API is available on Maven central alongside the latest JDBC drivers.

The Javadoc for the API is distributed alongside the Java library. An example of a simple Java application using the new Change Streams API is shown below:

```
import com.informix.jdbcx.IfxDataSource;
import com.informix.stream.api.IfmxStreamRecord;
import com.informix.stream.cdc.IfxCDCEngine;
import com.informix.stream.cdc.records.IfxCDCOperationRecord;

public class CDCExample {
  public static void main(String[] args) throws Exception {
    String url = args.length > 0 ? args[0]
        : "jdbc:informix-sqli://localhost:20290/syscdcv1:user=informix;password=informix";
    IfxDataSource ds = new IfxDataSource(url);
    IfxCDCEngine.Builder builder = new IfxCDCEngine.Builder(ds);
    builder.watchTable("testdb:informix.cdcTable", "a", "b");

    builder.timeout(5); // default 5 second timeout

    // Build the engine
    try (IfxCDCEngine engine = builder.build()) {

      // initialize the engine (creates the connections and begins listening for
      // changes)
      engine.init();
      IfmxStreamRecord record = null;

      // This loop is where you can inject logic that compiles
      // transactions, look for commits, throw away rollbacks
      // The data here is all Java typed, so it can be easily then
      // sent to MQTT, other JDBC drivers, streaming engines, or anything
      // else you can think of.
      while ((record = engine.getRecord()) != null) {
        // Print out the basic record information
        System.out.println(record);

        // If it is an insert/update/delete, print the column data
        if (record.hasOperationData()) {
          System.out.println(((IfxCDCOperationRecord) record).getData());
        }
      }
    }
  }
}
```

Index