Informix Product Family
Informix
Version 11.50

IBM Informix
Change Data Capture API
Programmer's Guide

IBM

Informix Product Family
Informix
Version 11.50

*IBM Informix
Change Data Capture API
Programmer's Guide*

IBM

Contents

Introduction

About this publication

This publication describes the IBM® Informix® Change Data Capture API and the concepts of capturing changes to data. This publication describes how to use the Change Data Capture API to write an application that captures changed data for external processing.

Types of users

This publication is for database application programmers.

To understand this publication, you must have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming in the C or Java programming language

What's new in the IBM Informix Change Data Capture API, Version 11.50

This publication includes information about new features and changes in existing functionality.

The following changes and enhancements are relevant to this publication. For a comprehensive list of all new features for this release, see the *IBM Informix Getting Started Guide*.

Table 1. What's New in IBM Informix Change Data Capture Programmer's Guide for Version 11.50.xC6

Overview	Reference
New Column Size Field Format for CDC Records The CDC_REC_DELETE, CDC_REC_INSERT, CDC_REC_UPDAFT, and CDC_REC_UPDBEF records have fields listing the size of each variable-length column in the row, if any. The column size fields in these CDC records are now 4 bytes long and in network byte, big-endian, order. Previously, the column size fields were the size of the operating system platform's integer byte length, in native byte order. To use the new column size field format, when you run the **cdc_opensess()** function, specify 1 as the value for both the *major_version* and *minor_version* arguments.	"The cdc_opensess() function" on page 2-5

Table 2. What's New in IBM Informix Change Data Capture Programmer's Guide for Version 11.50.xC5.

Overview	Reference
Monitor Change Data Capture Sessions You can now monitor the sessions involved in change data capture by using the new **onstat -g cdc** command. Use the options to display information about the captured tables, the buffers being used by sessions, the configuration of sessions, or the data capture activity.	Chapter 6, "onstat -g cdc," on page 6-1
View Change Data Capture Error Message Text You can now view the error message text corresponding to an error name by using the new cdc_errortext() function.	"The cdc_errortext() function" on page 2-4

Example code conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement. If you are using DB–Access, you must delimit multiple statements with semicolons.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the documentation for your product.

Additional documentation

Documentation about this release of IBM Informix products is available in various formats.

You can access or install the product documentation from the Quick Start CD that is shipped with Informix products. To get the most current information, see the Informix information centers at ibm.com®. You can access the information centers and other Informix technical information such as technotes, white papers, and IBM Redbooks® publications online at http://www.ibm.com/software/data/sw-library/.

Compliance with industry standards

IBM Informix products are compliant with various standards.

IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

The IBM Informix Geodetic DataBlade® Module supports a subset of the data types from the *Spatial Data Transfer Standard (SDTS)—Federal Information Processing Standard 173*, as referenced by the document *Content Standard for Geospatial Metadata*, Federal Geographic Data Committee, June 8, 1994 (FGDC Metadata Standard).

IBM Informix Dynamic Server (IDS) Enterprise Edition, Version 11.50 is certified under the Common Criteria. For more information, see *Common Criteria Certification: Requirements for IBM Informix Dynamic Server*, which is available at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss?CTY=US &FNC=SRX&PBL=SC23-7690-00.

Syntax diagrams

Syntax diagrams use special components to describe the syntax for statements and commands.

Table 3. Syntax Diagram Components

Component represented in PDF	Component represented in HTML	Meaning
▶▶———————————	`>>----------------------`	Statement begins.
——————————————▶	`---------------------->`	Statement continues on next line.
▶———————————	`>----------------------`	Statement continues from previous line.
——————————◀◀	`---------------------->< `	Statement ends.
———— SELECT ————	`--------SELECT----------`	Required item.
┌──────────┐ └ LOCAL ┘	`--+----------------+---` `  '------LOCAL------'`	Optional item.

── ALL ──
── DISTINCT──
── UNIQUE ──
	`---+-----ALL-------+---`
	`   +--DISTINCT-----+`
	`   '---UNIQUE------'`
		Required item with choice. Only one item must be present.

── FOR UPDATE ──
──FOR READ ONLY──
	`---+-----------------+---`
	`   +--FOR UPDATE-----+`
	`   '--FOR READ ONLY--'`
		Optional items with choice are shown below the main line, one of which you might specify.

Table 3. Syntax Diagram Components (continued)

Component represented in PDF	Component represented in HTML	Meaning

NEXT / PRIOR / PREVIOUS	`.---NEXT---------.`
	`----+---------------+---`
	`    +---PRIOR--------+`
	`    '---PREVIOUS-----'`
		The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line is used by default.

, / index_name / table_name	`.-------,----------.`
	`V                  |`
	`---+---------------+---`
	`   +---index_name---+`
	`   '---table_name---'`
		Optional items. Several items are allowed; a comma must precede each repetition.

►►─┤Table Reference├─►◄	`>>-| Table Reference |-><`	Reference to a syntax segment.

Table Reference
 	
view / table / synonym
	`Table Reference`
	`|--+-----view--------+--|`
	`   +------table------+`
	`   '----synonym------'`
		Syntax segment.

How to read a command-line syntax diagram

Command-line syntax diagrams use similar elements to those of other syntax diagrams.

Some of the elements are listed in the table in Syntax Diagrams.

Creating a no-conversion job

```
►►──onpladm create job─job──────────────── -n── -d─device── -D─database────────►
                        └─ -p─project─┘
```

```
►── -t─table────────────────────────────────────────────────────────────────►
```

```
►───────────────────────────────────────────────────────────►◄
   └─ -S─server─┘ └─ -T─target─┘  Setting the Run Mode  (1)
```
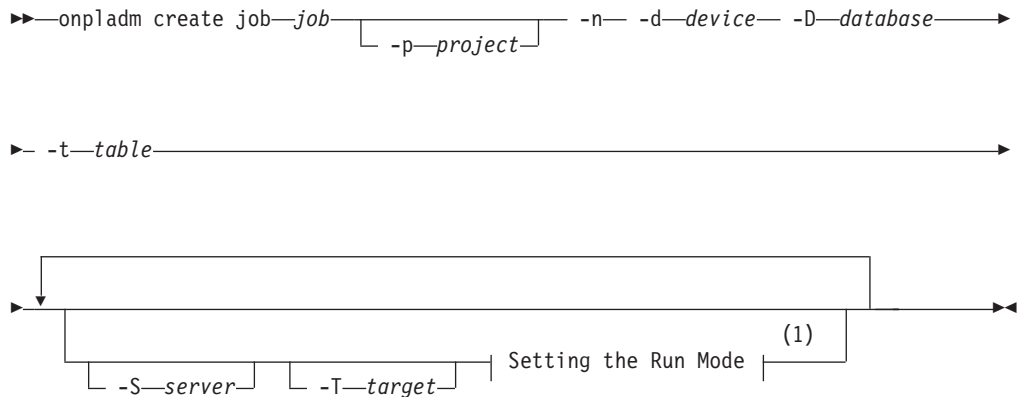
Notes:

1   See page Z-1

This diagram has a segment named "Setting the Run Mode," which according to the diagram footnote is on page Z-1. If this was an actual cross-reference, you would find this segment on the first page of Appendix Z. Instead, this segment is shown in the following segment diagram. Notice that the diagram uses segment start and end components.

Setting the run mode:

```
                  ┌─l─┐
                  │  └─c─┘
├── -f ──┬────┬───┼── u ──┬────┬──┬────┬─────────────────────┤
         ├─d─┤              └─n─┘  └─N─┘
         ├─p─┤
         └─a─┘
```

To see how to construct a command correctly, start at the upper left of the main diagram. Follow the diagram to the right, including the elements that you want. The elements in this diag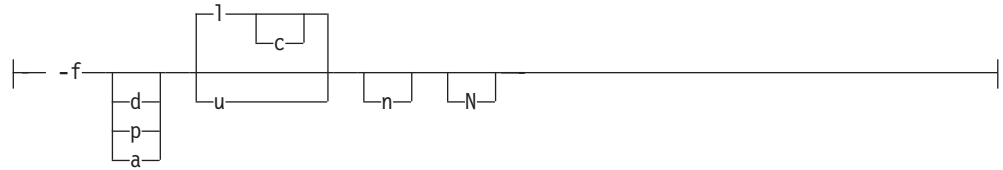ram are case-sensitive because they illustrate utility syntax. Other types of syntax, such as SQL, are not case-sensitive.

The Creating a No-Conversion Job diagram illustrates the following steps:
1. Type **onpladm create job** and then the name of the job.
2. Optionally, type **-p** and then the name of the project.
3. Type the following required elements:
   - **-n**
   - **-d** and the name of the device
   - **-D** and the name of the database
   - **-t** and the name of the table
4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
   - **-S** and the server name
   - **-T** and the target server name
   - The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type **-f**, optionally type **d**, **p**, or **a**, and then optionally type **l** or **u**.
5. Follow the diagram to the terminator.

Keywords and punctuation

Keywords are words reserved for statements and all commands except system-level commands.

When a keyword appears in a syntax diagram, it is shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

Identifiers and names

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples.

You can replace a variable with an arbitrary name, identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.

```
►►──SELECT──column_name──FROM──table_name────────────────────────►◄
```

When you write a SELECT statement of this form, you replace the variables *column_name* and *table_name* with the name of a specific column and table.

How to provide documentation feedback

You are encouraged to send your comments about IBM Informix user documentation.

Use one of the following methods:
- Send email to docinf@us.ibm.com.
- In the Informix information center, which is available online at http://www.ibm.com/software/data/sw-library/, open the topic that you want to comment on. Click the feedback link at the bottom of the page, fill out the form, and submit your feedback.
- Add comments to topics directly in the information center and read comments that were added by other users. Share information about the product documentation, participate in discussions with other users, rate topics, and more!

Feedback from all methods is monitored by the team that maintains the user documentation. The feedback methods are reserved for reporting errors and omissions in the documentation. For immediate help with a technical problem, contact IBM Technical Support. For instructions, see the IBM Informix Technical Support website at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Chapter 1. Getting started with the Change Data Capture API

These topics describe the Change Data Capture API and how to use it.

The Change Data Capture API

The Change Data Capture API allows external client applications to capture transactional data from an IBM Informix database.

The Change Data Capture 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 IBM 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 an IBM Informix database to another, heterogeneous, database.

The following types of operations are captured:
- INSERT
- DELETE
- UPDATE
- TRUNCATE

The Change Data Capture 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 Change Data Capture 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 Change Data Capture API.

The Change Data Capture 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 Change Data Capture API can capture those updates and populate the target table.

The Change Data Capture API does not capture changes to table schemas or any other database changes.

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

Change Data Capture API components

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

Functions

Change Data Capture 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 Change Data Capture 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 Change Data Capture functions and system tables. The system tables store information about Change Data Capture API error codes and record types.

Error codes

The Change Data Capture API functions return error codes. Most of the functions return an error code both if they succeed or fail. The Change Data Capture 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 Change Data Capture API; you can use smart large object read functions such as **mi_lo_read()** or **ifx_lo_read()**.

CDC Records

The Change Data Capture (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 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 1-1. Smart large object read functions

Read function	Arguments	Informix API	Application Language
mi_lo_read()	A pointer to a connection descriptor A smart large object file descriptor A data buffer The maximum number of bytes to read	DataBlade API	Use in a C language application.
ifx_lo_read()	A smart large object file descriptor A data buffer	ODBC	Use in an ODBC application.
ifx_lo_read()	A smart large object file descriptor A data buffer The maximum number of bytes to read A pointer to an error code	ESQL/C	Use in a C language application.
IfxLoRead()	A smart large object file descriptor A data buffer The maximum number of bytes to read	JDBC	Use in a Java application.
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 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 specified in the read call should 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.

The amount of data returned by a read call is the lesser of the size of the buffer 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 will be returned, even if the number of records returned is less than the maximum number allowed. If a record does not fit into a buffer, as much of the record as can fit is returned, and subsequent read calls return the rest.

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 IBM 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(char[] buff)
```

CDC record sequence numbers

Most Change Data Capture (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 IBM Informixlogical 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. A lower sequence number indicates that the CDC record was returned earlier than a CDC record with a higher sequence number.

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 IBM 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 Change Data Capture API, you must prepare the database and the database server.

Perform the following tasks to prepare for using the Change Data Capture API:

1. Turn on logging for all databases from which you intend to capture data changes. For information about logging, see the *IBM Informix Guide to SQL: Syntax*.
2. Run the following script as user **informix** from the $INFORMIXDIR/etc directory: 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 tasks

"Writing an application to capture data changes"

Writing an application to capture data changes

Use the Change Data Capture 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.

Complete the prerequisite tasks to prepare for using the Change Data Capture 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.

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. 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. 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 an 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 concepts

Chapter 2, "Change Data Capture functions," on page 2-1

Related tasks

"Preparing to use the Change Data Capture API" on page 1-5

Related reference

Chapter 7, "Change Data Capture sample program," on page 7-1

Handling errors

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

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 Change Data Capture error codes.
3. Set the error code variables to the Change Data Capture error numbers.
4. Add code to handle each error condition.

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

Related reference

Chapter 5, "Change Data Capture error codes," on page 5-1

Restarting data capture

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

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.

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. 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 reference

"The cdc_activatesess() function" on page 2-1

Monitoring data capture

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

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 reference

Chapter 6, "onstat -g cdc," on page 6-1

Chapter 2. Change Data Capture functions

These topics describe the Change Data Capture functions.

Related tasks

"Writing an application to capture data changes" on page 1-5

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 2-1. 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 tasks

"Restarting data capture" on page 1-6

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 2-2. 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 update to show that the capture session is not active.

Syntax

►►──cdc_deactivatesess──(──*session_ID*──)───────────────────────►◄

Function argument

Table 2-3. 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.

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 2-4. 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.

Table 2-4. The **cdc_endcapture()** *arguments (continued)*

Argument	Data Type	Description
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 2-5. 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.033'`.

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.033**.
- Other SQLCODES represent internal errors.

Example

The following example returns the error text for the error CDC_E_TABCAPTURED in the **en_us.033** locale:

```
> select cdc_errortext('CDC_E_TABCAPTURED', 'en_us.033') from syscdcvers;


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


1 row(s) retrieved.
```

Related reference

Chapter 5, "Change Data Capture error codes," on page 5-1

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 2-6. 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: **<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 Change Data Capture API. Must be 1.
minor_version	INTEGER	The minor version number of the Change Data Capture 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 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.

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 2-7. 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 2-8. 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: *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**. After you stop capturing data from a table, you can disable full-row logging.

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_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 2-9. **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.

Chapter 3. Change Data Capture records

These topics describe the CDC records returned from calls to read functions from an open capture session.

Format of CDC records

The Change Data Capture (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 3-1. 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.

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 3-2. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-3. 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.

Table 3-3. Format of the CDC_REC_COMMTX record (continued)

Section	Size	Description
Commit time	8 bytes	The UTC time at which the transaction was committed, in time_t format.

Related reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-4. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-5. 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.

Related reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-6. 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.
Error code	4 bytes	The error code.

Related reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-7. 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.

Table 3-7. The CDC_REC_INSERT record header (continued)

Section	Size	Description
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 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-8. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-9. 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 reference

"Format of CDC records" on page 3-1
"The syscdcrectypes table" on page 4-1

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 3-10. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-11. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-12. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

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 3-13. 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 reference

"Format of CDC records" on page 3-1

"The syscdcrectypes table" on page 4-1

Chapter 4. The syscdc system database

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

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

Table 4-1. 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 reference
Chapter 5, "Change Data Capture error codes," on page 5-1

The syscdcrectypes table

Contains the record types used by the Change Data Capture API.

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

Table 4-2. 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 reference

Chapter 5. Change Data Capture error codes

If a Change Data Capture 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 5-1. Change Data Capture 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.
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.

Table 5-1. Change Data Capture error codes (continued)

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

Related tasks

"Handling errors" on page 1-6

Related reference

"The syscdcerrcodes table" on page 4-1

"The cdc_errortext() function" on page 2-4

Chapter 6. onstat -g cdc

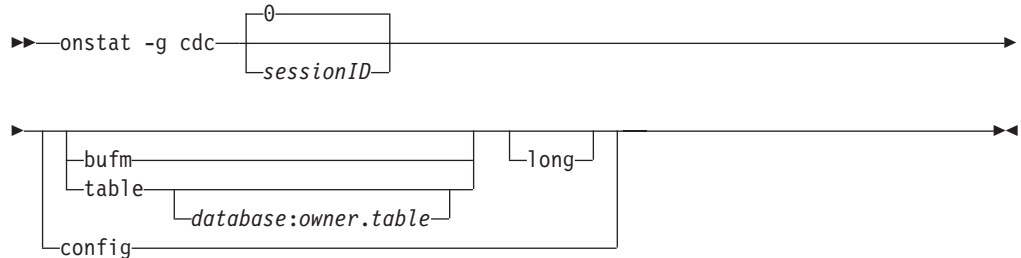Monitors the sessions involved in change data capture.

```
            0
►►─onstat -g cdc─┬─────────┬────────────────────────────────────────────►
                 └sessionID┘

►─┬──────────────────────────────────────┬──┬──────┬─────────────────────►◄
  ├─bufm─────────────────────────────────┤  └─long─┘
  ├─table─┬──────────────────────────┬───┤
  │       └─database:owner.table─────┘   │
  └─config───────────────────────────────┘
```

Table 6-1. 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: • 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 6-1. The **onstat -g cdc** *syntax elements (continued)*

Element	Purpose
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.

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 tasks

"Monitoring data capture" on page 1-7

Chapter 7. Change Data Capture sample program

Provides an example of using the Change Data Capture API to capture data and then processing that data.

The following sample program (version 10) creates an application that captures data from multiple tables. The program runs Change Data Capture 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 allows the user to enter the database name, the table name, column names, and the timeout value.

This program requires that the **getopt** and **getsubopt** parser functions are implemented on your computer.

```
/**************************************************************************
 *
 * Licensed Materials - Property of Informix
 *
 * Informix
 * (c) Copyright IBM Corporation 2008, 2010 All rights reserved.
 *
 *  Title:         cdcapi.ec
 *
 *  Description:
 *
 *      Example program that uses the CDC log capture API.
 *
 *      This program starts capture on one or more tables.
 * Command line options are:
 *  -D <database_name where table exists>
 *  -T <Table name Format : "owner.tabname">
 *  -C <Column name>
 *  -t <timeout in seconds>
 *      -R <max records per read>
 *  -L <LSN to start >
 *
 * e.g
 *    cdcapi -D db1 -T user1.t1 -C "a,b" -T t2 -C "a,c"
 *
 * The program exits when it receives a timeout record.
 *
 * To compile:
 * On UNIX:
 *  esql -g -static -o cdcapi
 *    cdcapi.ec $INFORMIXDIR/lib/dmi/libdmi.a
 * On Windows:
 *  esql -g -static -o cdcapi.exe
 *    cdcapi.ec %INFORMIXDIR%\lib\dmi\libthdmi.lib
 *
 **************************************************************************
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <memory.h>
#include <ctype.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <stdarg.h>
#include <assert.h>

#ifdef _WIN32
#include <windows.h>
#define sleep(x) Sleep(x*1000)
#else
#include <unistd.h>
#endif

EXEC SQL include     sqlca.h ;
EXEC SQL include     sqltypes;
EXEC SQL include     sqlca;
EXEC SQL include     sqlda;
EXEC SQL include     decimal;

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#ifndef SQLINFXBIGINT
#define SQLINFXBIGINT  SQLBIGINT
#endif

/* varchar length includes 1 byte to hold
 * the length of the column data
 */
#define VARCHAR_LEN_OFFSET      1

/* lvarchar length includes 3 bytes to hold
 * the length of the column data
 */
#define LVARCHAR_LEN_OFFSET     3

/* Int8 data has 2byte for sign, 4byte for lower int
 * and 4 byte for higher int
 */
#define INT8_LO_OFFSET          2
#define INT8_HI_OFFSET          6

/* LOB handle offset */
#define LOHANDLE_OFFSET         4

/* BOOL col size */
#define BOOL_COL_LEN    2



/* Display length of the column values */
#define DISPLAY_LEN             31



#define BYTESHIFT 8
#define BYTEMASK 0xFF

#define ld2(p) ((int4)(((p)[0]<<BYTESHIFT)+((p)[1]&BYTEMASK)))


int isLittleEndian = 0;
```

```c
void checkPlatformEndianess()
{
  short  i;
  char *a=(char*)&i;

  a[0]=0; a[1]=1;

  if ( i == 1)
       isLittleEndian = 0;
  else
       isLittleEndian = 1;


}

/* double and float comes in bigendian format, we need routines to
 * extract them correctly depending upon then platform on which  this program
 * runs.
 */
double mylddbl(const char *p)
{
    double fval;

    if (isLittleEndian)
        {
        char *f;
        int c;
        int i = 0;
        f = (char *) &fval + sizeof(double)-1;
    c = sizeof(double);
    do
        *f-- = *p++;
    while (--c);
        }
  else
       memcpy((char*)&fval,p,sizeof(double));

    return(fval);
}

float myldfloat(const char *p)
{
    float fval;

    if (isLittleEndian)
        {
        char *f;
        int c;
        int i = 0;
        f = (char *) &fval + sizeof(float)-1;
    c = sizeof(float);
    do
        *f-- = *p++;
    while (--c);
        }
  else
       memcpy((char*)&fval,p,sizeof(float));

    return(fval);
}

/* List of CDC_ log records this program processes */
typedef enum {
 CDC_REC_BEGINTX=0,
 CDC_REC_COMMTX,
 CDC_REC_RBTX,
 CDC_REC_INSERT,
```

```
 CDC_REC_DELETE,
 CDC_REC_UPDBEF,
 CDC_REC_UPDAFT,
 CDC_REC_DISCARD,
 CDC_REC_TRUNCATE,
 CDC_REC_TABSCHEMA,
 CDC_REC_TIMEOUT,
 CDC_REC_ERROR,
 NUM_CDCLOGRECTYPES
} cdc_log_type_t;

struct cdc_logrec
{
 cdc_log_type_t rectype;
 char recname[17];
} ;

typedef struct cdc_logrec cdc_logrec_t;

/* Table of the CDC_ log rectypes and their symbolic names as
 * published in the Manual
 */
cdc_logrec_t  logrectab[NUM_CDCLOGRECTYPES] =
{
 {CDC_REC_BEGINTX, "CDC_REC_BEGINTX"},
 {CDC_REC_COMMTX, "CDC_REC_COMMTX"},
 {CDC_REC_RBTX,  "CDC_REC_RBTX"},
 {CDC_REC_INSERT, "CDC_REC_INSERT"},
 {CDC_REC_DELETE, "CDC_REC_DELETE"},
 {CDC_REC_UPDBEF, "CDC_REC_UPDBEF"},
 {CDC_REC_UPDAFT, "CDC_REC_UPDAFT"},
 {CDC_REC_DISCARD, "CDC_REC_DISCARD"},
 {CDC_REC_TRUNCATE, "CDC_REC_TRUNCATE"},
 {CDC_REC_TABSCHEMA, "CDC_REC_TABSCHEM"},
 {CDC_REC_TIMEOUT, "CDC_REC_TIMEOUT"},
 {CDC_REC_ERROR,  "CDC_REC_ERROR"}

};

/* Structures to represent and store data from syscdcrectype table */
struct _syscdcrectypes
{
 int recnum;
 char recname[17];
 char recdesc[128];
};

typedef struct _syscdcrectypes syscdcrectypes_t;
syscdcrectypes_t  *syscdcrectypes_tab;
int num_cdc_log_recs;

/* Structures to represent and store data from syscdcerrorcode table */
struct _syscdcerrcode
{
 int errcode;
 char errname[17];
 char errdesc[128];
};

typedef struct _syscdcerrcode syscdcerrcode_t;
syscdcerrcode_t  *syscdcerrcode_tab;
int num_cdc_err_recs;

char    outfile[512]={0};
FILE    *outfilefp = NULL;
```

```c
void  get_cdc_rectype_name(int recnum, char* recsymb, char* recdesc);
cdc_log_type_t get_cdc_log_type(char* recsymb);
void  get_cdc_error_info(int cdcerr, char* cdcerrsymb, char* cdcerrdesc);
void  loadSyscdcrectypes(void);
void  loadSyscdcerrorcodes(void);

int num_of_var_col(int);
void  get_col_types(int, char *);
void get_col_values(int, char *);
void set_num_of_var_col(int, int);
void close_session(int);
void dump_hex(char *, char *, int);
void sw_CheckSQLCode(char *, char *, int);
void sw_SQLExecImmed(char *, char *, int);
void  get_lob_data(char* dbuf);
void  printStdoutAndFile(char*  msg, ...);

/* Common Header size of a CDC record */
typedef struct
    {
    int4 ch_size_hdr;
    int4 ch_size_payload;
    int4 ch_payload_type;
    } common_header_t;

#define CDC_CMNHDR_SIZE (sizeof(common_header_t))
#define CDC_LOGRECTYPE_SIZE   4


/* Structures for getting column type information for the captured tables */
typedef struct column_obj {
 int  coltype;
 int     colxid;
 int  colsize;
 char *colname;
} column_obj_t;

#define MAXCOL 50
typedef struct column_desc {
    int   num_of_columns;
    int   num_of_vchar;
    column_obj_t colobj[MAXCOL];
} column_des_t;

#define  MAXTAB 5
column_des_t        g_coldesc[MAXTAB];

/*  Use full macros */
#define CHK_SQL_CODE(message) \
       sw_CheckSQLCode(message, __FILE__, __LINE__)

#define SQL_EXEC_I(command) \
       sw_SQLExecImmed(command, __FILE__, __LINE__)

#define CHK_CDC_RETVAL(message,cdenum) \
       sw_checkCDCRetval(message,cdenum, __FILE__, __LINE__)

#ifndef MAX
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
#endif
#ifndef MIN
#define MIN(a,b) (((a) < (b)) ? (a) : (b))
#endif

/* Rest of the Global  variables */
int nodebug=0;
```

```
EXEC SQL begin declare section;
EXEC SQL Define NAMELEN  100;
EXEC SQL Define MAXARGSIZE 512;
char dbname[NAMELEN];
char svrname[NAMELEN];
char arg1[MAXARGSIZE];
char arg2[MAXARGSIZE];
int sessionid = 0;
int timeout = 300;
int max_recs_per_read = 1;
int bytes_per_read = 64*1024;
bigint lsn;
EXEC SQL end declare section;

int      remainderBytes = 0;

/* Test related variables */
char    datafile[512]={0};
char    system_str[1024]={0};
char    schema_expected[2048]={0};
char    *schema_str;
int     schema_fix_bytes=0;
int     schema_no_fix_cols=0;
int     schema_no_var_cols=0;
int     caperrcode=0;
int     CmpSchema=0;


/* On Windows, ldbigint is not called correctly. So this is a workaround */

#ifdef _WIN32
#define BYTESHFTx 8
#define BYTEMASKx 0xFF

void ldbigintx(bigint *bint, char *p)
{
    *bint = ((((((((((((((((bigint) p[0]
        << BYTESHFTx) + (p[1] & BYTEMASKx))
        << BYTESHFTx) + (p[2] & BYTEMASKx))
        << BYTESHFTx) + (p[3] & BYTEMASKx))
        << BYTESHFTx) + (p[4] & BYTEMASKx))
        << BYTESHFTx) + (p[5] & BYTEMASKx))
        << BYTESHFTx) + (p[6] & BYTEMASKx))
        << BYTESHFTx) + (p[7] & BYTEMASKx)));
}
#else
#define ldbigintx ldbigint
#endif

void
Usage(char *progname)
{

 fprintf(stdout,"%s -D <db> [-t <timeout>] [-R <max records per read>] [-L <start lsn>] [-f outfile] \n",
  progname);
 fprintf(stdout,"\t list of tables and col list -T <tab1> -C <col list> \n");
 fprintf(stdout,"\nCommand Line Options are :\n");
 fprintf(stdout,"\t-D <database_name where table exists>\n");
 fprintf(stdout,"\t-t <timeout in second>\n");
 fprintf(stdout,"\t\tDefault: 300 sec\n");
 fprintf(stdout,"\t-R <max records per read>\n");
 fprintf(stdout,"\t\tDefault: 1\n");
 fprintf(stdout,"\t-L <LSN to start capture. Format \"<uniq id>:0x<pos>\">\n" );
 fprintf(stdout,"\t\tDefault: current LSN\n");
 fprintf(stdout,"\t-T <Name of the table to be captured. Format \"owner.tabname\">\n");
 fprintf(stdout,"\t-C <List of Columns to be captured in format \"c1,c2\"  >\n");
 fprintf(stdout,"\t-f <Name of the file to dump the CDC log records>\n");
```

```c
 fprintf(stdout,"\nProgram Description\n");
 fprintf(stdout,"\tThis program starts capture on one or more tables.\n");
 fprintf(stdout,"\tProgram terminates when timeout record is received.\n");
 fprintf(stdout,"\n\tExample:\n");
 fprintf(stdout,"\t  %s -D db1 -T \"bill.t1\" -C \"a,b\" -T t2 -C \"a,c\"\n",progname);
exit(0);
}

int
num_of_var_column(int tabid)
{
    return g_coldesc[tabid].num_of_vchar;
}

void
set_num_of_var_col(int tabid, int no)
{
    g_coldesc[tabid].num_of_vchar = no;
}

void
sw_SQLExecImmed(command, fname, lnum)
EXEC SQL begin declare section;
char    *command;
EXEC SQL end declare section;
int     lnum;
char    *fname;
{
    if (nodebug != 1)
     printf("%s at %s:%d\n", command, fname, lnum);
    EXEC SQL execute immediate :command;
    sw_CheckSQLCode(command, fname, lnum);
}


void
sw_CheckSQLCode(char *msgptr, char *fname, int lnum)
{
    if(SQLCODE < 0)
 {
    printf("\nStatment %s Failed at %s:%d. SQLCODE = %d ISAM = %d",
     msgptr, fname, lnum, SQLCODE,(long)sqlca.sqlerrd[1]);

 close_session(0);
 }
}

void
sw_checkCDCRetval(char *msgptr,  int cdcenum,  char *fname, int lnum)
{
 char cdcerrdesc[128];
 char cdcerrcode[17];

 if (cdcenum < 0)
  {
  get_cdc_error_info(cdcenum,cdcerrcode,cdcerrdesc);
     printf("\nCDC API '%s' Failed at %s:%d.",
         msgptr, fname, lnum);
  printf("\n\tCDCAPI_RETVAL = %d,  CDC_ERRORCODE = %s",
          cdcenum, cdcerrcode);
   printf("\nCDC Error Description is \n\t%s\n", cdcerrdesc);
  }
 sw_CheckSQLCode(msgptr,fname,lnum);
}

EXEC SQL Define SQL_STMT_LEN 8192;
EXEC SQL DEFINE IDS_MAX_LEN 1024;
```

```
void
get_col_values(int tabid, char *databuf)
{
    int     col, len, total_len, host_len, nullflag, maxlen;
    short   c_smallint;
    int     c_integer;
    float   c_float;
    double  c_double;
    dec_t   c_decimal;
    dtime_t   c_datetime;
    char    dec_str[DISPLAY_LEN+1], data_ptr[DISPLAY_LEN+1];
    char    vlen;
    ifx_sqlvar_t  *colp;
char   *vcharlen_array;
int    vcharlen=0;
    int     collen;
    column_des_t coldesc = g_coldesc[tabid];
    bigint   c_bigint;
    ifx_int8_t  c_int8;
$boolean  c_boolean;

    /* Part of the initial data which holds size of variable length col */
    vcharlen_array = (databuf - coldesc.num_of_vchar * sizeof(int));

    for (col=0; col < coldesc.num_of_columns; col++)
{
    int advanced_to_next_col = FALSE;
memset(data_ptr,0,DISPLAY_LEN);

switch (MASKNONULL (coldesc.colobj[col].coltype)) {
    case SQLINT8 :
    case SQLSERIAL8:
  c_int8.sign = ld2(databuf);
  c_int8.data[0] = ldlong(databuf+INT8_LO_OFFSET);
  c_int8.data[1] = ldlong(databuf+INT8_HI_OFFSET);

  if (risnull(CINT8TYPE,(char*)&c_int8))
   sprintf(data_ptr, "NULL");
  else
   {
   ifx_int8toasc(&c_int8, data_ptr, DISPLAY_LEN);
   data_ptr[DISPLAY_LEN] = '\0';
   }
  printStdoutAndFile("\tColumn Value = %s\n", data_ptr);
 break;
    case SQLSERIAL :
    case SQLDATE :
    case SQLINT :
  c_integer = ldlong(databuf);
  if (risnull(CINTTYPE,(char*)&c_integer))
   printStdoutAndFile("\tColumn Value = %s\n", "NULL");
  else
   printStdoutAndFile("\tColumn Value = %d\n", c_integer);
 break;
    case SQLBOOL :
  /* boolean stream has 2 bytes, first byte indicate nullness
            * second byte indicate 1 (true) or 0 (false)
   */
  if (*databuf == 1)
   printStdoutAndFile("\tColumn Value = %s\n", "NULL");
  else
   {
   c_boolean = *(databuf+1);
   printStdoutAndFile("\tColumn Value = %d\n", c_boolean);
   }
            advanced_to_next_col = TRUE;
```

```
            databuf += BOOL_COL_LEN;
break;
    case SQLCHAR :
 if (risnull(CCHARTYPE,databuf))
  sprintf(data_ptr, "NULL");
 else
  {
  maxlen = MIN(coldesc.colobj[col].colsize, DISPLAY_LEN);
        memcpy(data_ptr, databuf, maxlen);
  data_ptr[maxlen] = '\0';
  }
 printStdoutAndFile("\tColumn Value = '%s'\n", data_ptr);
break;
    case SQLNVCHAR :
    case SQLVCHAR :
          vcharlen = ldlong(vcharlen_array);

 collen = vcharlen - VARCHAR_LEN_OFFSET;
 maxlen = MIN(collen, DISPLAY_LEN);
 if (risnull(CVCHARTYPE, (databuf+VARCHAR_LEN_OFFSET)))
  sprintf(data_ptr, "NULL");
 else
  {
        memcpy(data_ptr, databuf+VARCHAR_LEN_OFFSET, maxlen);
  data_ptr[maxlen] = '\0';
  }
 printStdoutAndFile("\tColumn Value = '%s'\n", data_ptr);

          advanced_to_next_col = TRUE;
          databuf += collen+VARCHAR_LEN_OFFSET;
 vcharlen_array += 4;
break;
    case SQLINFXBIGINT :
 ldbigintx(&c_bigint, databuf);
 if (risnull(CBIGINTTYPE,(char*)&c_bigint))
  printStdoutAndFile("\tColumn Value = %s\n", "NULL");
 else
          printStdoutAndFile("\tColumn Value = %lld\n", c_bigint);
break;
    case SQLFLOAT :
 c_double = mylddbl(databuf);
 if (risnull(CDOUBLETYPE,(char*)&c_double))
  printStdoutAndFile("\tColumn Value = %s\n", "NULL");
 else
  printStdoutAndFile("\tColumn Value = %lf\n", c_double);
break;
    case SQLSMFLOAT :
 c_float = myldfloat(databuf);
 if (risnull(CFLOATTYPE,(char*)&c_float))
  printStdoutAndFile("\tColumn Value = %s\n", "NULL");
 else
  printStdoutAndFile("\tColumn Value = %f\n", c_float);
break;
    case SQLSMINT :
 c_smallint = ld2(databuf);
 if (risnull(CSHORTTYPE,(char*)&c_smallint))
  printStdoutAndFile("\tColumn Value = %s\n", "NULL");
 else
  printStdoutAndFile("\tColumn Value = %hd\n", c_smallint);
break;
    case SQLMONEY :
    case SQLDECIMAL :
 lddecimal(databuf, coldesc.colobj[col].colsize, &c_decimal);
 if (risnull(CDECIMALTYPE,(char*)&c_decimal))
  printStdoutAndFile("\tColumn Value = %s\n", "NULL");
 else
  {
```

```
       dectoasc(&c_decimal, data_ptr,
        DISPLAY_LEN,  coldesc.colobj[col].colsize);
       data_ptr[DISPLAY_LEN]='\0';
       printStdoutAndFile("\tColumn Value = %s\n", data_ptr);
       }
  break;
       case SQLDTIME:
       case SQLINTERVAL:
    lddecimal(databuf, coldesc.colobj[col].colsize,
       &(c_datetime.dt_dec));
    if (risnull(CDTIMETYPE,(char*)&(c_datetime.dt_dec)))
     printStdoutAndFile("\tColumn Value = %s\n", "NULL");
     else
      {
         memset (dec_str, 0, DISPLAY_LEN+1);
               dectoasc (&c_datetime.dt_dec, data_ptr, DISPLAY_LEN,
          coldesc.colobj[col].colsize);
      data_ptr[DISPLAY_LEN]='\0';
      printStdoutAndFile("\tColumn Value = %s\n", data_ptr);
      }
  break;
       case SQLLVARCHAR :

               vcharlen = ldlong(vcharlen_array);

    collen = vcharlen - LVARCHAR_LEN_OFFSET;
    maxlen = MIN(collen, DISPLAY_LEN);
    if (risnull(CLVCHARTYPE,databuf+LVARCHAR_LEN_OFFSET))
     sprintf(data_ptr, "NULL");
     else
      {
          memcpy(data_ptr, databuf+LVARCHAR_LEN_OFFSET, maxlen);
      data_ptr[maxlen] = '\0';
      }
    printStdoutAndFile("\tColumn Value = '%s'\n", data_ptr);

               advanced_to_next_col = TRUE;

     databuf += collen + LVARCHAR_LEN_OFFSET;

    vcharlen_array += 4;
   break;
   case SQLUDTFIXED :
               if ((coldesc.colobj[col].colxid == XID_BLOB) ||
                   (coldesc.colobj[col].colxid == XID_CLOB))
               {
               get_lob_data(databuf);
               databuf += LOHANDLE_OFFSET;
               }
         else
             printf("Could not find data type %d xid %d\n",
                 coldesc.colobj[col].coltype,
                 coldesc.colobj[col].colxid);
         break;

       default:
    printf("Could not find data type %d\n",
       coldesc.colobj[col].coltype);
   break;
       }

     /*
      * Advance to the next column.  Some cases already do this explicitly.
      */
     if (! advanced_to_next_col)
         {
         databuf += coldesc.colobj[col].colsize;
```

```
        }
 }

}

void
get_col_types(int tabid, char *colstring)
{
    $char       sql_stm[SQL_STMT_LEN];
    char      tabname[IDS_MAX_LEN];
    ifx_sqlda_t   *sqlda;
    int      col, colsize;
    column_des_t   coldesc;

    sprintf(tabname, "apitest%d%d", tabid, getpid());
    sprintf(sql_stm, "create temp table %s( %s )",
    tabname, colstring);
    SQL_EXEC_I(sql_stm);

    sprintf(sql_stm, "select * from %s", tabname);
    $prepare select_id from $sql_stm;
    CHK_SQL_CODE(sql_stm);

    $describe select_id into sqlda;
    CHK_SQL_CODE("Describe");

    for (col = 0; col < sqlda->sqld; col++)
  {
  colsize = rtypsize(sqlda->sqlvar[col].sqltype,
      sqlda->sqlvar[col].sqllen);
  printStdoutAndFile("\tColumn %d is %s, type = %d, size = %d\n", col,
      sqlda->sqlvar[col].sqlname, sqlda->sqlvar[col].sqltype, colsize);

      coldesc.colobj[col].coltype = sqlda->sqlvar[col].sqltype;
  coldesc.colobj[col].colsize = colsize;
  coldesc.colobj[col].colxid = sqlda->sqlvar[col].sqlxid;
  coldesc.colobj[col].colname =
   malloc(strlen(sqlda->sqlvar[col].sqlname)+1);
  strcpy(coldesc.colobj[col].colname, sqlda->sqlvar[col].sqlname);
  }
 coldesc.num_of_columns = col;

    g_coldesc[tabid] = coldesc;

    sprintf(sql_stm, "drop table %s", tabname);
    SQL_EXEC_I(sql_stm);
}


void
close_session(int return_from_fun)
{
    $integer retval;

    if (sessionid > 0)
        {
        fprintf(stdout, "\nCDC_CLOSESESS on session %d\n", sessionid);
        $execute function informix.cdc_closesess(:sessionid)
                    into :retval;
      CHK_CDC_RETVAL("cdc_closesess",retval);
        }

    if(return_from_fun)
   return;
    else
  {
  if (outfilefp)
```

```
            fclose(outfilefp);
        exit(caperrcode);
    }
}

/* Various field positions of a CDC_REC_TRUNCATE log record */
#define TRUNCATE_LSN_HI_OFFSET          0
#define TRUNCATE_LSN_LO_OFFSET          (TRUNCATE_LSN_HI_OFFSET + 4)
#define TRUNCATE_TXNID_OFFSET           (TRUNCATE_LSN_LO_OFFSET + 4)
#define TRUNCATE_USERDATA_OFFSET        (TRUNCATE_TXNID_OFFSET + 4)


void
process_truncate(char *databuf, int size)
{
    int     lsn_lo;
    int     lsn_hi;
    int     txid;
    char    *insertdata;
    int     tabid = 0;

    lsn_hi = ldlong(databuf + TRUNCATE_LSN_HI_OFFSET);
    lsn_lo = ldlong(databuf + TRUNCATE_LSN_LO_OFFSET);
    printf("LSN = %d:0x%x. ", lsn_hi, lsn_lo);

    txid = ldlong(databuf + TRUNCATE_TXNID_OFFSET);
    printf("TXID = %d", txid);
    printStdoutAndFile("\n");

    tabid = ldlong(databuf + TRUNCATE_USERDATA_OFFSET);
    printf("tabid = %d\n", tabid);
}


/* Various field offsets of a CDC_REC_TABSCHEMA  record */
#define TABSCHEMA_USERDATA_OFFSET        0
#define TABSCHEMA_RESERVED_OFFSET        (TABSCHEMA_USERDATA_OFFSET + 4)
#define TABSCHEMA_FIXEDCOLWIDTH_OFFSET   (TABSCHEMA_RESERVED_OFFSET + 4)
#define TABSCHEMA_NUMFIXEDCOL_OFFSET     (TABSCHEMA_FIXEDCOLWIDTH_OFFSET + 4)
#define TABSCHEMA_NUMVARCOL_OFFSET       (TABSCHEMA_NUMFIXEDCOL_OFFSET + 4)
#define TABSCHEMA_COLDESC_OFFSET         (TABSCHEMA_NUMVARCOL_OFFSET + 4)

void
process_tabschema(char *databuf)
{
 char *coldesc_data;
 int fixed_col_width;
 int no_of_fixed_col;
 int no_of_var_col;
    int tabid = 0;

    /* First 4 bytes are userdata */
    tabid = ldlong(databuf+TABSCHEMA_USERDATA_OFFSET);
    printf("\tTabID : %d", tabid);
    printStdoutAndFile("\n");

    /* The 4 bytes following first 8 bytes is total size of fixed column */
    fixed_col_width = ldlong(databuf+TABSCHEMA_FIXEDCOLWIDTH_OFFSET);
    printf("\tFixed length column size (total) : %d\n", fixed_col_width);

 /* Test related changes */
 if ((CmpSchema) && fixed_col_width != schema_fix_bytes)
     {
        printf("\tFAILED, Expected fixed col bytes is %d\n", schema_fix_bytes);
        caperrcode++;
        }
```

```c
    /* Next 4 bytes is no of fixed length column */
    no_of_fixed_col = ldlong(databuf + TABSCHEMA_NUMFIXEDCOL_OFFSET);
    printf("\tNumber of fixed lenth column is %d\n", no_of_fixed_col);

 /* Test related changes */
 if ((CmpSchema) && no_of_fixed_col != schema_no_fix_cols)
        {
        printf("\tFAILED, Expected fixed col numbers is %d\n",
      schema_no_fix_cols);
        caperrcode++;
        }



    /* Next 4 bytes is no of variable length column */
    no_of_var_col = ldlong(databuf + TABSCHEMA_NUMVARCOL_OFFSET);
    printf("\tNumber of variable lenth column is %d\n", no_of_var_col);

 /* Test related changes */
 if ((CmpSchema) && no_of_var_col != schema_no_var_cols)
        {
        printf("\tFAILED, Expected variable col numbers is %d\n",
      schema_no_var_cols);
        caperrcode++;
        }



    /* Rest is column description string */
    coldesc_data = databuf + TABSCHEMA_COLDESC_OFFSET;
    printStdoutAndFile("\tSchema for Table is: '%s'\n", coldesc_data);

 /* Test related changes */
 if ((CmpSchema) && strcmp(coldesc_data, schema_str))
        {
        printf("\tExpected Schema for Table is: '%s'\n", schema_str);
        printf("\tSchema doesn't match: FAILED\n");
        caperrcode++;
        }
    else
        printf("\tSchema matchs: PASSED\n");



    get_col_types(tabid, coldesc_data);
    set_num_of_var_col(tabid, no_of_var_col);
}

char *
FmtUTCTime(char *buf, time_t t)
{
    struct tm   *ltime = localtime(&t);

    sprintf(buf, "%04u-%02u-%02u %02u:%02u:%02u",
            (1900 + ltime->tm_year)%10000, ltime->tm_mon + 1, ltime->tm_mday,
            ltime->tm_hour, ltime->tm_min, ltime->tm_sec);
    return buf;
}

/* Various field offsets of a Transactional  records
 *  CDC_REC_BEGINTX
 *  CDC_REC_COMMTX
 *  CDC_REC_RBTX
 */
#define TXN_LSN_HI_OFFSET           0
#define TXN_LSN_LO_OFFSET           (TXN_LSN_HI_OFFSET + 4)
#define TXN_TXNID_OFFSET            (TXN_LSN_LO_OFFSET + 4)
```

```
#define TXN_USERDATA_OFFSET          (TXN_TXNID_OFFSET + 4)
#define TXN_TIME_OFFSET              (TXN_USERDATA_OFFSET + 4)

void
process_TXN_Records(char *databuf, int size)
{
    int  lsn_lo;
    int  lsn_hi;
    int  txid;
 int  longtime;
    int  tabid = 0;
    char timebuf[512];

    lsn_hi = ldlong(databuf + TXN_LSN_HI_OFFSET);
    lsn_lo = ldlong(databuf + TXN_LSN_LO_OFFSET);
    printf("LSN = %d:0x%x. ", lsn_hi, lsn_lo);

    txid = ldlong(databuf + TXN_TXNID_OFFSET);
    printf("TXID = %d", txid);
    printStdoutAndFile("\n");

    tabid = ldlong(databuf + TXN_USERDATA_OFFSET);

    longtime = ldlong(databuf + TXN_TIME_OFFSET);
    FmtUTCTime(timebuf,(time_t)longtime);
    printf("\tTime = %s\n", timebuf);

    dump_hex("Raw Data: ", databuf, size);
}

/* Various field offsets of a CDC_REC_DISCARD log record */
#define DISCARD_LSN_HI_OFFSET          0
#define DISCARD_LSN_LO_OFFSET          (DISCARD_LSN_HI_OFFSET + 4)
#define DISCARD_TXNID_OFFSET           (DISCARD_LSN_LO_OFFSET + 4)
#define DISCARD_USERDATA_OFFSET        (DISCARD_TXNID_OFFSET + 4)


void
process_discard(char *databuf, int size)
{
    int  lsn_lo;
    int  lsn_hi;
    int  txid;
    int  tabid = 0;

    dump_hex("Raw Data: ", databuf, 3);
    printf("\tTotal record size = %d\n", size);

    lsn_hi = ldlong(databuf + DISCARD_LSN_HI_OFFSET);
    lsn_lo = ldlong(databuf + DISCARD_LSN_LO_OFFSET);
    printf("\tLSN = %d:0x%x\n", lsn_hi, lsn_lo);

    txid = ldlong(databuf + DISCARD_TXNID_OFFSET);
    printf("\tTXID = %d", txid);
    printStdoutAndFile("\n");

    tabid = ldlong(databuf + DISCARD_USERDATA_OFFSET);
    dump_hex("Raw Data: ", databuf, size);
}


/* Various field offsets of a IUD (INSERT/UPDATE/DELETE)  records
 *  CDC_REC_INSERT,
 *  CDC_REC_DELETE,
 *  CDC_REC_UPDBEF,
 *  CDC_REC_UPDAFT,
 */
```

```
#define IUD_LSN_HI_OFFSET          0
#define IUD_LSN_LO_OFFSET          (IUD_LSN_HI_OFFSET + 4)
#define IUD_TXNID_OFFSET           (IUD_LSN_LO_OFFSET + 4)
#define IUD_USERDATA_OFFSET        (IUD_TXNID_OFFSET + 4)
#define IUD_FLAG_OFFSET            (IUD_USERDATA_OFFSET + 4)
/* For every variable length column there is 4 bytes for length */
#define IUD_ROWDATA_OFFSET(nvarchars)      (IUD_FLAG_OFFSET + 4 + \
                                                (nvarchars * 4))


void
process_IUD_Records(char *databuf, int size)
{
    int  lsn_lo;
    int  lsn_hi;
    int  txid;
    char *insertdata;
    int  tabid = 0;
 int   numvarchars=0;

    lsn_hi = ldlong(databuf + IUD_LSN_HI_OFFSET);
    lsn_lo = ldlong(databuf + IUD_LSN_LO_OFFSET);
    printf("LSN = %d:0x%x. ", lsn_hi, lsn_lo);

    txid = ldlong(databuf + IUD_TXNID_OFFSET);
    printf("TXID = %d", txid);
    printStdoutAndFile("\n");

    tabid = ldlong(databuf + IUD_USERDATA_OFFSET);
    printf("\tTabID = %d\n", tabid);

 numvarchars = num_of_var_column(tabid);
    insertdata = databuf+IUD_ROWDATA_OFFSET(numvarchars);
    dump_hex("Raw IUD Data: ", insertdata, size);

 /* Now we are pointing to the beginning of the column values
     * Extract values for each of the columns and print them after
     * formatting to the correct type
     */
    get_col_values(tabid, insertdata);
}

int
isIUDRecord(cdc_log_type_t logrec)
{
 switch(logrec)
 {
 case CDC_REC_INSERT:
        case CDC_REC_DELETE:
        case CDC_REC_UPDBEF:
        case CDC_REC_UPDAFT:
   return 1;
        break;
  default:
   return 0;
 }
}

cdc_log_type_t   cur_logrec_type;
char    *cur_record_ptr;
common_header_t  cur_log_header;


/*
 ****************************************************************************
 * process_record()
 *
```

```
 * Description:
 *      Process a single record.
 *
 * Input:
 *     databuf       Points to start of record, the entire record must be in
 *                   the databuf.
 * Outputs:
 *      o_bytes_processed    How many bytes were processed.
 *
 * Returns:
 *      0 all ok, complete record processed
 *      2 timeout occurred
 *      < 0 error
 ******************************************************************************
 */
mint
process_record(char* a_recordbuf, int a_bytes_in_buf, int *o_bytes_processed)
{
    int  logrectype;
    char recsymb[17], recdesc[128];
    int bytes_in_this_rec = 0;

    *o_bytes_processed = 0;

    /* Get the common header information from the CDC packet*/
    cur_log_header.ch_size_hdr   = ldlong(a_recordbuf);
    cur_log_header.ch_size_payload  = ldlong(a_recordbuf+4);
    cur_log_header.ch_payload_type  = ldlong(a_recordbuf+8);

    bytes_in_this_rec = cur_log_header.ch_size_hdr +
                                    cur_log_header.ch_size_payload;

    if (a_bytes_in_buf < bytes_in_this_rec)
        {
        fprintf(stderr,
                "process_record: Too few bytes in recordbuf: %d need %d\n",
                 a_bytes_in_buf, bytes_in_this_rec);

        return -1;
        }

    /* Check what we got is a correct CDC packet.
     * 66 is the value of CDC_PKTSCHEME_LRECBINARY in the
     * syscdcpacketschemes table.
     */
    if (cur_log_header.ch_payload_type != 66)
        {
        fprintf(stderr, "Unknown packet type: 0x%x\n",
                        cur_log_header.ch_payload_type);
        return -1;
        }

    /*
     * The 4 bytes following the common header are the log record type.
     */
    logrectype = ldlong(a_recordbuf + CDC_CMNHDR_SIZE);

    /*
     * Set cur_record_ptr to point to immediately after the packet type
     */
    cur_record_ptr = a_recordbuf + (CDC_CMNHDR_SIZE + CDC_LOGRECTYPE_SIZE);


    /* Get the symbolic name and description for this log rec type */
    get_cdc_rectype_name(logrectype, recsymb,recdesc);
    printStdoutAndFile("Got Record type %17s. Size = %4d ",
                    recsymb,cur_log_header.ch_size_payload);
```

```
    /* Map the logrectype to the type that this program is interested */
    cur_logrec_type = get_cdc_log_type(recsymb);

    /*
     * Now switch on the log record type and extract  data from it.
    */
    switch (cur_logrec_type)
         {
    case CDC_REC_BEGINTX:
      process_TXN_Records(cur_record_ptr, cur_log_header.ch_size_payload);
        break;
  case CDC_REC_COMMTX:
      process_TXN_Records(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_RBTX:
       process_TXN_Records(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_INSERT:
       process_IUD_Records(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_DELETE:
       process_IUD_Records(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_UPDBEF:
       process_IUD_Records(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_UPDAFT:
       process_IUD_Records(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_DISCARD:
       process_discard(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_TRUNCATE:
   process_truncate(cur_record_ptr, cur_log_header.ch_size_payload);
      break;
  case CDC_REC_TABSCHEMA:
       process_tabschema(cur_record_ptr);
      break;
  case CDC_REC_TIMEOUT:
               *o_bytes_processed = bytes_in_this_rec;
    return 2;
    break;

  case CDC_REC_ERROR:
   /* Not handling this record in this program */
      break;

  case NUM_CDCLOGRECTYPES:
     default:
  break;
         }
    *o_bytes_processed = bytes_in_this_rec;

 return 0;
}




#define MAXTAB 5

int no_of_tab = -1;
char dbname[NAMELEN];
char tabname[MAXTAB][NAMELEN];
char columns[MAXTAB][10*NAMELEN];
char tab_schema[MAXTAB][NAMELEN];
```

```c
void
init_default(char *progname)
{
    strcpy(dbname, "foo");
    strcpy(tabname[0], "informix.t1");
    strcpy(columns[0], "aa,bb");
    lsn = 0LL;
}


#define QUESTION_MARK    '?'
int optind = 1;
int opterr = 1;
int optopt;
char *optarg;

void
ProcessOption(int ac, char *av[])
{
    int         opt, lsn1, lsn2;
    int         ignore_err;
    char lsn_string[1024];
 char*       token;

    if (ac <= 1)
 Usage(av[0]);

    init_default(av[0]);

    while ((opt = getopt(ac, av, "C:D:L:R:t:T:h:n:f:S:d:")) != EOF)
        switch(opt) {
        case 'C' :
            strncpy(columns[no_of_tab], optarg, 10*NAMELEN);
            break;
        case 'D' :
            strncpy(dbname, optarg, NAMELEN);
            break;
    case 'L' :
  strncpy(lsn_string, optarg, 1024);
 sscanf(lsn_string, "%d:%x", &lsn1, &lsn2);
     lsn = ((((bigint)lsn1 << 32)) | (bigint)lsn2);
  printf("LSN to start snoop(%d:0x%x = %lld\n",
    lsn1, lsn2, lsn);
  break;
        case 'R':
  max_recs_per_read = atoi(optarg);
           break;
    case 't':
  timeout = atoi(optarg);
  break;
        case 'T' :
  no_of_tab++;
            strncpy(tabname[no_of_tab], optarg, NAMELEN);
            break;
    case 'n' :
            nodebug = 1;
  break;
  case 'f' :
            strncpy(outfile, optarg, NAMELEN);
            outfilefp = fopen(outfile, "w");
            if (!outfilefp)
                {
                printf("file %s open failed\n", outfile);
                exit(-1);
                }
         break;
  case 'S':
```

```
   CmpSchema = 1;
   strncpy(schema_expected, optarg, 2000);
   token = strtok(schema_expected, ":");
   schema_fix_bytes = atoi(token);
   token = strtok(NULL, ":");
   schema_no_fix_cols = atoi(token);
   token = strtok(NULL, ":");
   schema_no_var_cols = atoi(token);
   schema_str = strtok(NULL, ":");
   break;
 case 'd':
   strncpy(datafile, optarg, NAMELEN);
   break;

    case '?' :
    case 'h' :
              Usage(av[0]);
   break;
 }
}


int
main(int argc, char **argv)
{
    char   *progname;
    char*   databuf = NULL;
    int    capture_started = 0;
    int    i, endofread=0;
    char   *servername;
 int    status;
    int         stat_bufs_read = 0;          /* statistics - how many buffers */
    int         stat_recs_extracted = 0;    /* statistics - how many records */
    int         bytes_left_over_in_previous_buf = 0;
    $integer retval, tabid;

    progname = argv[0];

    checkPlatformEndianess();

 /* Process the arguments passed to this program */
    ProcessOption(argc, argv);
    if(no_of_tab < 0)
 no_of_tab = 0;

    databuf = malloc(128*1024);
    if (! databuf)
        {
 fprintf(stderr, "%s: Malloc failed\n", progname);
        exit (-3);
        }

 /* Setup the connection to syscdcv1 database against the
  * current Informix instance
  */
    servername = getenv("INFORMIXSERVER");
    if(servername != NULL)
     sprintf(arg1, servername);
    else
 {
 fprintf(stderr, "INFORMIXSERVER env variable not set.\n");
 exit (-1);
 }

    $connect to "syscdcv1";
    CHK_SQL_CODE("connect to syscdcv1");
    fprintf(stdout, "Connected to syscdcv1@%s\n", servername);
```

```
   retval = 0;

/* Load the syscdcrectypes and syscdcerrcodes table rows */
loadSyscdcrectypes();
loadSyscdcerrorcodes();

/* Start a CDC session */
   fprintf(stdout,
       "CDC_OPENSESS for server %s with  Timeout %d Max recs per read %d\n",
   arg1,timeout, max_recs_per_read);
   $execute function informix.cdc_opensess(:arg1, 0,
 :timeout, :max_recs_per_read, 1, 1) into :retval;
   CHK_CDC_RETVAL("cdc_opensess",retval);

/* Save the CDC session id */
   sessionid = retval;

   if (retval < 0)
       goto done;

   for(i=0; i <= no_of_tab; i++)
   {
   sprintf(arg1, "%s:%s", dbname, tabname[i]);

/* Enable FULL ROW LOGGING for the table */
   fprintf(stdout, "Enable for full row logging on Table '%s'\n", arg1);
   $execute function informix.cdc_set_fullrowlogging(:arg1, 1) into :retval;
   CHK_CDC_RETVAL("cdc_set_fullrowlogging",retval);
   if (retval < 0)
       goto done;

/* Start the capture of the table */
   fprintf(stdout, "CDC_STARTCAPTURE of %s on session %d\n", arg1,sessionid);
   sprintf(arg2, "%s", columns[i]);
   tabid = i;
   $execute function informix.cdc_startcapture(:sessionid, 0,
               :arg1, :arg2, :tabid)
               into :retval;
   CHK_CDC_RETVAL("cdc_startcapture",retval);
   if (retval < 0)
       goto done;
   capture_started = 1;
   }

/* Activate the CDC session */
   fprintf(stdout, "CDC_ACTIVATESESS on  %d \n", sessionid);
   $execute function informix.cdc_activatesess(:sessionid, :lsn)
               into :retval;
   CHK_CDC_RETVAL("cdc_activatesess",retval);
   if (retval < 0)
       goto done;

/* Test related changes */
if (datafile[0] != '\0')
 {
 sleep(5);
 fprintf(stdout, "invoking dbaccess to run %s\n", datafile);
 sprintf(system_str, "dbaccess - %s > %s.out  2>&1", datafile,datafile);
 system(system_str);
 system("onmode -c");
 }


   fprintf(stdout, "Start Reading the log records...\n");
   bytes_left_over_in_previous_buf = 0;
```

```
   while (!endofread )
       {
       int bytesread;
       int loreaderr = 0;
       char* recptr;              /* start of a record */

       /*
        * If the previous iteration split a record, copy what we had to
        * the beginning of the buffer so we get the tail end.
        */
       if (bytes_left_over_in_previous_buf > 0)
           {
           memcpy(databuf, recptr, bytes_left_over_in_previous_buf);
           }

       recptr = databuf;

       /*
        * Read more data into the buffer
        */
       bytesread = ifx_lo_read(sessionid,
                               &databuf[bytes_left_over_in_previous_buf],
                                   bytes_per_read, &loreaderr);
 if (nodebug != 1)
       fprintf(stdout, "bytesread is %d loreaderr is %d SQLCODE %d\n",
                     bytesread, loreaderr, SQLCODE);

    if(loreaderr < 0)
           {
           fprintf(stderr, "%s: loreaderr %d SQLCODE %d\n",
                             progname, loreaderr, SQLCODE);
           printStdoutAndFile("%s: ifx_lo_read loreaderr %d SQLCODE %d\n",
                             progname, loreaderr, SQLCODE);
    goto done;
           }

     stat_bufs_read ++;

     /*
      * Process each record in the buffer.
      */
 if (bytesread > 0)
 {
         int bytes_in_buf;

         bytes_in_buf = bytesread + bytes_left_over_in_previous_buf;
         bytes_left_over_in_previous_buf = 0;

         /*
          * Process each full record in the databuf.  recptr always
          * points to the start of a record.
          */

         while (bytes_in_buf > 0)
             {
             int4 hdrsize, payloadsize;

             hdrsize = ldlong(recptr);
             payloadsize = ldlong(recptr+4);

             if (bytes_in_buf >= hdrsize + payloadsize)
                 {
                 int recstatus;
                 int bytes_processed;

                 recstatus = process_record(recptr, bytes_in_buf,
                                             &bytes_processed);
```

```
                    if (recstatus < 0)
                        {
                        goto extraction_error;
                        }

                    assert(bytes_processed > 0);
                    stat_recs_extracted++;

                    recptr += bytes_processed;
                    bytes_in_buf -= bytes_processed;

                    if (recstatus == 2)
                        {
                        goto timeout_occurred;
                        }

                    }
                else
                    {
                    bytes_left_over_in_previous_buf = bytes_in_buf;
                    bytes_in_buf = 0;
                    }
                }
            }
        else
            {
            fprintf(stderr, "%s: ifx_lo_read returned 0 bytes read. Exiting\n",
                    progname);
            goto done;
            }
        }

timeout_occurred:

    fprintf(stdout, "\nGoing to End the CDC session....\n");

done:
    if (retval < 0)
        caperrcode ++;

    if (capture_started)
    {
    for(i=0; i <= no_of_tab; i++)
     {
     sprintf(arg1, "%s:%s", dbname, tabname[i]);
        fprintf(stdout, "CDC_ENDCAPTURE of '%s' on session %d\n",
       arg1,sessionid);
        $execute function informix.cdc_endcapture(:sessionid, 0,
                          :arg1)
                          into :retval;
    CHK_CDC_RETVAL("cdc_endcapture",retval);

    fprintf(stdout, "Disable full row logging on Table %s\n", arg1);
    $execute function informix.cdc_set_fullrowlogging(:arg1, 0)
            into :retval;

    CHK_CDC_RETVAL("cdc_endcapture",retval);
    }
     }
    if (sessionid > 0)
        {
        fprintf(stdout, "cdc_closesess on session %d\n", sessionid);
        $execute function informix.cdc_closesess(:sessionid)
                  into :retval;
    CHK_CDC_RETVAL("cdc_closesess",retval);
        }
```

```
    if (databuf)
        free(databuf);

    if (nodebug != 1)
        {
        /* Doesn't like ?: construct inside the printf arg list
         * so do it like this*/
        float avg_recs;
        if (stat_bufs_read == 0)
            avg_recs = 0.0;
        else
            avg_recs = ((float) stat_recs_extracted) / ((float) stat_bufs_read);

        fprintf(stdout, "Total buffers read: %d\n", stat_bufs_read);
        fprintf(stdout, "Total records extracted: %d\n", stat_recs_extracted);
        fprintf(stdout, "Average recs per buffer: %f\n", avg_recs);
        }

    exit(caperrcode);

extraction_error:
    fprintf(stderr, "%s: Record extraction failed.\n", progname);
    goto done;
} /* main() */

#define PRINTABLE(c) (isprint(c) ? (c) : '.')

void
dump_hex (char *str, char *rec, int len)
{
  int i, j;

  if (nodebug==1) return;

  fprintf(stdout,"%s (%d/0x%x bytes at address 0x%x)\n",
    str, len, len, rec);


  if (rec == NULL || len < 1)
    return;

  for (i = 0; i < len; i += 16)
  {
    for (j = i; j < i + 16; j++)
    {
      if (j < len)
                printStdoutAndFile("%02x ", rec[j] & 0xff);
      else
        printStdoutAndFile ("   ");
    }
    for (j = i; j < i + 16; j++)
      if (j < len)
        printStdoutAndFile ("%c", PRINTABLE(rec[j] & 0xff));
    printStdoutAndFile ("\n");
  }
}

/* Extract the BLOB and CLOB data from the LO Header information of log rec*/
void
get_lob_data(char* dbuf)
{
  int err;
  ifx_lo_t    LO={0};
  $int lofd;
  $ifx_lo_stat_t *stats;
  $ifx_int8_t size;
```

```
   int isize;
   char *lobuff ;

do {
    printStdoutAndFile("\tColumn Value = <SBLOB Data>\n");
    /* Skip the SQL Null indicator in the first 4 bytes */
    memcpy((char*)&LO, dbuf+4, sizeof(ifx_lo_t));

    $set lock mode to wait;

    lofd = ifx_lo_open(&LO, LO_RDONLY, &err);
 if (lofd < 0)
 {
        printStdoutAndFile("\tSBLOB Data is NULL on open %d\n", err);
 break;
 }
     if(ifx_lo_stat(lofd, &stats) < 0)
        {
        printStdoutAndFile("\tSBLOB Data is NULL on stat\n");
 break;
        }
    if((ifx_lo_stat_size(stats, &size)) < 0)
            isize = 0;
    else if(ifx_int8toint(&size, &isize) !=  0)
        {
        fprintf(stdout,"\nFailed to convert size");
        isize = 0;
        break;
        }
    printStdoutAndFile("\tSBLOB size is %d\n",isize);

    lobuff = malloc(isize);

    isize = ifx_lo_read(lofd, lobuff, isize, &err);
    if (isize < 0)
        {
        fprintf(stdout,"\t cant read LO %d\n", err);
  CHK_SQL_CODE("cant read LO");
        break;
        }

    printStdoutAndFile("\tifx_lo_read output  size=%d \n",isize);
    dump_hex("SBLOB Data:", lobuff, isize);
    } while(0);


    /* Close smart large object */
    ifx_lo_close(lofd);
}

void
printStdoutAndFile(char*  msg, ...)
{
   va_list ap;
 char bigbuf[256];

    va_start(ap, msg);
    vsprintf(bigbuf,msg,ap);
 va_end(ap);

 fprintf(stdout,"%s",bigbuf);
 if (outfilefp)
  {
  fprintf(outfilefp,"%s",bigbuf);
  fflush(outfilefp);
  }
 return;
```

```
}

/* For a given CDC log record type that we snooped, find the symbolic name
 * and the description
 */
void
get_cdc_rectype_name(int i_recnum, char* o_recsymb, char* o_recdesc)
{
 int i ;
 for (i=0; i < num_cdc_log_recs; i++)
  {
  if (syscdcrectypes_tab[i].recnum == i_recnum)
   {
   if (o_recsymb)
    strcpy(o_recsymb, syscdcrectypes_tab[i].recname);
   if (o_recdesc)
     strcpy(o_recdesc,syscdcrectypes_tab[i].recdesc);
   return;
   }
  }

 if (o_recsymb)
  strcpy(o_recsymb, "UNKNOWN");
 if (o_recdesc)
   strcpy(o_recdesc, "Unknown/UnDocumented Log record");
}

/* For a given symbolic name of the CDC log record find the logrec type
 * that this program defined
 */
cdc_log_type_t
get_cdc_log_type(char* i_recname)
{
 int i;
 for (i=0; i < NUM_CDCLOGRECTYPES; i++)
 {
  if (strcmp(i_recname, logrectab[i].recname) == 0)
   return logrectab[i].rectype;
 }

 printf("Error: Did not find value for %s log rec symbol table\n",i_recname);
 return NUM_CDCLOGRECTYPES;
}

/* Find detailed CDC error message and CDC error symbolic name for a given
 * error code returned by the CDC APIs
 */
void
get_cdc_error_info(int i_cdcerr, char* o_cdcerrsymb, char* o_cdcerrdesc)
{
 int i ;
 for (i=0; i < num_cdc_err_recs; i++)
  {
  if (syscdcerrcode_tab[i].errcode == i_cdcerr)
   {
   if (o_cdcerrsymb)
    strcpy(o_cdcerrsymb, syscdcerrcode_tab[i].errname);
   if (o_cdcerrdesc)
    {
    $lvarchar* lvmsgp;
    $char* errnamep;
    char* msgtextp;

    errnamep = syscdcerrcode_tab[i].errname;

                /* Obtain the message in the default locale ('' SQL NULL as
                 * the second argument to cdc_errortext).  We can also
```

```
                   * use a specific locale name such as 'en_US.033' as the
                   * second argument to the routine.
      *
      * Use an lvarchar rather than char[] variable to hold the
      * function return value so that the message text is not
      * blank-padded.
      */

    ifx_var_flag(&lvmsgp, 1);
    $execute function informix.cdc_errortext(:errnamep, '')
      into :lvmsgp;
    msgtextp = (char*) ifx_var_getdata(&lvmsgp);
    if (msgtextp)
      {
      strcpy(o_cdcerrdesc, msgtextp);
      }
    else
      {
      /* Fall back to the message comment. */
      strcpy(o_cdcerrdesc,syscdcerrcode_tab[i].errdesc);
      }

    ifx_var_dealloc(&lvmsgp);

    }
   return;
   }
  }

 if (o_cdcerrsymb)
  strcpy(o_cdcerrsymb, "UNKNOWN");
 if (o_cdcerrdesc)
   strcpy(o_cdcerrdesc, "Unknown/Undocumented CDC API Error");

}


/* Query the syscdcrectype and store it in an in-memory table */
void
loadSyscdcrectypes(void)
{
 $int sys_recnum;
 $varchar sys_recname[17];
 $varchar sys_recdesc[128];
 $int numrec=0;
 int i=0;

 EXEC SQL select count(*) into :numrec from syscdcrectypes;

 if (numrec <= 0)
  {
  num_cdc_log_recs=0;
  return;
  }

 syscdcrectypes_tab = (syscdcrectypes_t*) malloc(numrec *
     sizeof(syscdcrectypes_t));

 num_cdc_log_recs=numrec;
 EXEC SQL DECLARE rectype_cur CURSOR FOR
     SELECT recnum, recname, recdesc
        INTO :sys_recnum, :sys_recname, :sys_recdesc
        FROM syscdcrectypes
  WHERE recname LIKE "CDC%"
     FOR READ ONLY;

 EXEC SQL OPEN rectype_cur;
```

```
  while(SQLCODE == 0)
  {
     EXEC SQL FETCH rectype_cur;
     if(SQLCODE == 0)
   {
  sys_recname[16]='\0';
  sys_recdesc[127]='\0';
  syscdcrectypes_tab[i].recnum = sys_recnum;
  strcpy(syscdcrectypes_tab[i].recname, sys_recname);
  strcpy(syscdcrectypes_tab[i].recdesc, sys_recdesc);
  i++;
   }
  }

  EXEC SQL CLOSE rectype_cur;
}

/* Query the syscdcerrocdoes and store it in an in-memory table */
void
loadSyscdcerrorcodes(void)
{
 $int sys_errcode;
 $varchar sys_errname[16];
 $varchar sys_errdesc[128];
 $int enumrec=0;
 int i=0;

 EXEC SQL select count(*) into :enumrec from syscdcerrcodes;

 if (enumrec <= 0)
  {
  num_cdc_err_recs=0;
  return;
  }

 syscdcerrcode_tab = (syscdcerrcode_t*) malloc(enumrec *
     sizeof(syscdcerrcode_t));

 num_cdc_err_recs=enumrec;
 EXEC SQL DECLARE errcode_cur CURSOR FOR
     SELECT errcode, errname, errdesc
        INTO :sys_errcode, :sys_errname, :sys_errdesc
        FROM syscdcerrcodes
     FOR READ ONLY;

 EXEC SQL OPEN errcode_cur;

 while(SQLCODE == 0)
 {
    EXEC SQL FETCH errcode_cur;
    if(SQLCODE == 0)
  {
  syscdcerrcode_tab[i].errcode = sys_errcode;
  strcpy(syscdcerrcode_tab[i].errname, sys_errname);
  strcpy(syscdcerrcode_tab[i].errdesc, sys_errdesc);
  i++;
  }
 }

 EXEC SQL CLOSE errcode_cur;
}
```

The following sample output illustrates this program.

```
/u/informix> ./cdcapi  -D test -T informix.t1 -t 30  -C "col1,col2,col3" -f x.out
Connected to syscdcv1@new1
CDC_OPENSESS for server new1 with  Timeout 30
Enable for full row logging on Table 'test:informix.t1'
CDC_STARTCAPTURE of test:informix.t1 on session 39
CDC_ACTIVATESESS on  39
Start Reading the log records...
Got Record type  CDC_REC_TABSCHEM. Size =   37  TabID : 0
        Fixed length column size (total) : 15
        Number of fixed lenth column is 3
        Number of variable lenth column is 0
        Schema for Table is: 'col1 serial, col2 char(1), col3 int8'
        Column 0 is col1, type = 6, size = 4
        Column 1 is col2, type = 0, size = 1
        Column 2 is col3, type = 17, size = 10
Got Record type   CDC_REC_BEGINTX. Size =    0 LSN = 47:0x5e018. TXID = 24
        Time = 2008-10-23 19:01:51
Got Record type    CDC_REC_INSERT. Size =   15 LSN = 47:0x5e04c. TXID = 24
        TabID = 0
        Column Value = 1
        Column Value = 'a'
        Column Value = 1000
Got Record type    CDC_REC_COMMTX. Size =    0 LSN = 47:0x5e09c. TXID = 24
        Time = 2008-10-23 19:01:51
...
...

Got Record type   CDC_REC_TIMEOUT. Size =    0
Going to End the CDC session....
CDC_ENDCAPTURE of 'test:informix.t1' on session 39
Disable full row logging on Table test:informix.t1
cdc_closesess on session 39
```

Related tasks

"Writing an application to capture data changes" on page 1-5

Appendix. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix products

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in IBM Informix products. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Tip: The information center and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features by using the keyboard instead of the mouse.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

Related accessibility information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software.

You can view the publications in Adobe Portable Document Format (PDF) by using the Adobe Acrobat Reader.

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the IBM commitment to accessibility.

Dotted decimal syntax diagrams

The syntax diagrams in our publications are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader.

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive

alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, that element is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 refers to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

?   Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

!   Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used.

However, if you hear the lines 2? `FILE`, 2.1, 2.1.1! `(KEEP)`, and 2.1.1 `(DELETE)`, the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? `FILE`. Nothing is used if the keyword `FILE` is omitted.

*   Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* `data-area`, you know that you can include more than one data area or you can include none. If you hear the lines 3*, 3 `HOST`, and 3 `STATE`, you know that you can include `HOST`, `STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.

2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.

3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

\+   Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line 6.1+ `data-area`, you must include at least one data area. If you hear the lines 2+, 2 `HOST`, and 2 `STATE`, you know that you must include `HOST`, `STATE`, or both. As for the * symbol, you can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy,

modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

IBM®

Printed in USA