Oracle® Database

XStream Guide

11*g* Release 2 (11.2)

E16545-07

August 2011

ORACLE®

Oracle Database XStream Guide, 11*g* Release 2 (11.2)

E16545-07

Primary Author: Randy Urbano

Contributors: Lance Ashdown, Vinoth Chandar, Alan Downing, Thuvan Hoang, Richard Huang, Joydip Kundu, Tianshu Li, Edwina Lu, Rui Mao, Pat McElroy, Valarie Moore, Srikanth Nalla, Partha Raghunathan, Ashish Ray, Jim Stamos, Byron Wang, Lik Wong, Jun Yuan, Rod Ward, Haobo Xu, Kevin Xu

Contents

Preface

Oracle Database XStream Guide describes the features and functionality of XStream. This document contains conceptual information about XStream, along with information about configuring and managing an XStream environment. In addition, this document contains reference information related to XStream.

Audience

This guide is intended for database administrators who configure and manage XStream environments. To use this document, database administrators must be familiar with relational database concepts, SQL, distributed database administration, Oracle Streams concepts, PL/SQL, and the operating systems under which they run an XStream environment.

This guide is also intended for programmers who develop applications that use XStream. To use this document, programmers need knowledge of an application development language and relational database concepts.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Database XStream Java API Reference*
- *Oracle Streams Concepts and Administration*
- *Oracle Streams Replication Administrator's Guide*
- *Oracle Call Interface Programmer's Guide*
- *Oracle Database 2 Day + Java Developer's Guide*

- *Oracle Database Java Developer's Guide*
- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database SQL Language Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Database PL/SQL Language Reference*
- *Oracle Streams Advanced Queuing User's Guide*

Many of the examples in this book use the sample schemas of the sample database, which is installed by default when you install Oracle Database. Refer to *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them yourself.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

XStream Concepts and Use Cases

This part contains the following chapters:

1 Introduction to XStream

This chapter introduces you to XStream, a new feature in Oracle Database 11*g* Release 2 (11.2). XStream enables information sharing with outstanding performance and usability.

This chapter contains the following topics:

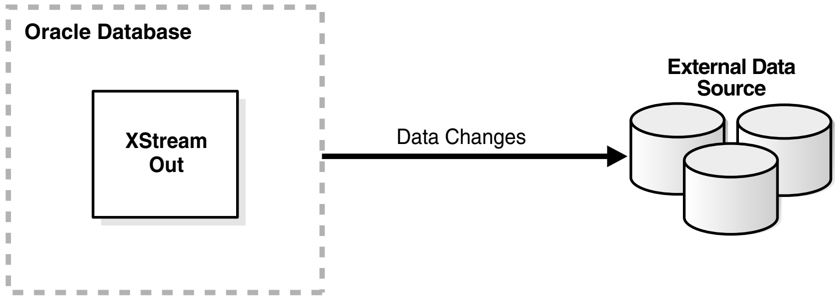
- About XStream
- Purpose of XStream
- Prerequisites for XStream
- Tasks and Tools for XStream

About XStream

XStream consists of Oracle Database components and application programming interfaces (APIs) that enable client applications to receive data changes from an Oracle database and send data changes to an Oracle database. These data changes can be shared between Oracle databases and other systems. The other systems include non-Oracle databases, non-RDBMS Oracle products, file systems, third party software applications, and so on. A client application is designed by the user for specific purposes and use cases.

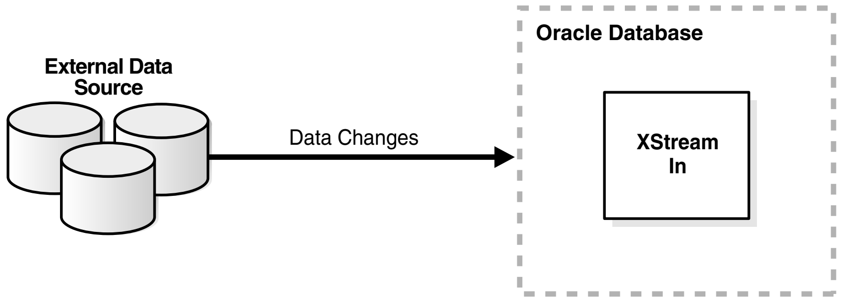
XStream consists of two major features: XStream Out and XStream In. **XStream Out** provides Oracle Database components and APIs that enable you to share data changes made to an Oracle database with other systems.

Figure 1–1 XStream Out



XStream In provides Oracle Database components and APIs that enable you to share data changes made to other systems with an Oracle database.

Figure 1–2 XStream In



XStream is built on the infrastructure of Oracle Streams. Therefore, XStream inherits the flexibility and functionality of Oracle Streams, including:

- The logical change record (LCR) format for streaming database changes

  An **LCR** is a message with a specific format that describes a database change. If the change was a data manipulation language (DML) operation, then a **row LCR** encapsulates each row change resulting from the DML operation. One DML operation might result in multiple row changes, and so one DML operation might result in multiple row LCRs. If the change was a data definition language (DDL) operation, then a single **DDL LCR** encapsulates the DDL change.
- Filtering of database changes at the database level, schema level, table level, and row/column level
- Rules and rule sets that control behavior, including inclusion and exclusion rules
- Rule-based transformations that modify captured data changes
- Support for the data types supported by Oracle Streams, including LOBs, `LONG`, `LONG RAW`, and `XMLType`
- Customized configurations, including multiple inbound streams to a single database instance, multiple outbound streams from a single database instance, multiple outbound streams from a single capture process, and so on
- Full-featured apply for XStream In, including apply parallelism for optimal performance, SQL generation, conflict detection and resolution, error handling, and customized apply with apply handlers

---

Note: When learning about and using XStream, a general knowledge of Oracle Streams concepts is helpful. See the following documents for conceptual information about Oracle Streams:

- *Oracle Database 2 Day + Data Replication and Integration Guide* contains basic conceptual information about Oracle Streams
- *Oracle Streams Concepts and Administration* contains detailed conceptual information about Oracle Streams

---

See Also:

- Chapter 4, "Configuring XStream"
- Chapter 8, "DBMS_XSTREAM_ADM"

Purpose of XStream

By using XStream, you can accomplish the following goals:

- Replicate data changes

  Replication is generally used to improve availability and to improve performance by spreading the network load over multiple regions and servers. XStream enables you replicate data changes made to an Oracle database with other Oracle databases and with non-Oracle data sources.

- Store data changes in files

  Some environments use files to store data changes for various reasons. For example, an environment might use files to store data changes if the environment does not have a physical network, if the environment uses disconnected computing, or if the environment uses satellite communications. After data changes are stored in files, the changes can be processed in any customized way by applications.

- Share data changes with a client-side memory cache

  Some environments share data changes with a client-side memory cache to improve performance.

See Also: Chapter 3, "XStream Use Cases"

Prerequisites for XStream

Using the XStream APIs requires purchasing a license for the Oracle GoldenGate product. See the documentation for the Oracle GoldenGate product for more information:

`http://download.oracle.com/docs/cd/E15881_01/index.htm`

In addition, this document assumes that you have the following skills:

- Knowledge of relational database concepts and Oracle Database concepts

  XStream includes components that run in an Oracle database. To use XStream successfully, you must be able to administer an Oracle Database.

  See *Oracle Database Concepts* for information about this topic.

- Knowledge of distributed databases

  An XStream environment includes multiple data sources, including Oracle databases and non-Oracle data sources. You should understand distributed database concepts before using XStream.

  See *Oracle Database Administrator's Guide* for information about this topic.

- Knowledge of SQL and PL/SQL

  To administer an Oracle database and the XStream components running in an Oracle database, you must know how to use SQL and PL/SQL.

  See *Oracle Database SQL Language Reference*, *Oracle Database PL/SQL Language Reference*, and *Oracle Database PL/SQL Packages and Types Reference* for information about this topic.

- Knowledge of application programming

  XStream Out sends data changes to a client application for processing. XStream In receives data changes from a client application. You use the Oracle Call Interface

(OCI) API or the Java API to create a client application that communicates with XStream.

See *Oracle Call Interface Programmer's Guide* for information about the OCI API.

See *Oracle Database 2 Day + Java Developer's Guide* and *Oracle Database Java Developer's Guide* for information about the Java API.

Tasks and Tools for XStream

This section describes the common tasks you perform for XStream and the tools to use to complete the tasks.

This section contains the following topics:

- XStream Tasks
- XStream Tools

XStream Tasks

The common tasks for XStream are the following:

- Configure XStream

  Configuring XStream involves preparing an Oracle Database for XStream, creating the Oracle Database components used by XStream, and creating one or more client applications that communicate with the Oracle Database.

  See Chapter 4, "Configuring XStream" for information about this task.

- Administer XStream

  Administering XStream involves managing the Oracle Database components used by XStream. It also involves managing the rules and rule sets used by these components. It might also require modifications to a client application.

  See Chapter 5, "Managing XStream" for information about this task.

- Monitor XStream

  Monitoring XStream involves viewing Oracle Enterprise Manager pages related to XStream and querying data dictionary views related to XStream.

  See the Oracle Enterprise Manager online help and Chapter 6, "Monitoring XStream" for information about this task.

XStream Tools

Use the following tools to complete the tasks for XStream:

- SQL and PL/SQL

  You can use SQL and PL/SQL to configure, administer, and monitor XStream. SQL enables you to create an XStream administrator and monitor XStream using data dictionary views. Several Oracle-supplied PL/SQL packages enable you to configure and manage XStream.

  See *Oracle Database SQL Language Reference*, *Oracle Database PL/SQL Language Reference*, and *Oracle Database PL/SQL Packages and Types Reference* for information about this topic.

- Oracle Enterprise Manager

You can use Enterprise Manager to manage and monitor XStream components. You can also use Enterprise Manager to view information about the LCRs that are streaming in an XStream configuration.

See the Enterprise Manager online help for more information about this topic.

- The OCI API and Java API

  You can use the XStream OCI API and XStream Java API to create client application that communicate with XStream. These applications can work with XStream Out to stream LCRs out of an Oracle Database, and these applications can work with XStream In to stream LCRs into an Oracle Database.

  See Part IV, "XStream OCI API Reference" for information about the XStream OCI API, and see *Oracle Call Interface Programmer's Guide* for information about the OCI API.

  See *Oracle Database XStream Java API Reference* for information about the XStream Java API, and see *Oracle Database 2 Day + Java Developer's Guide* and *Oracle Database Java Developer's Guide* for information about the Java API.

2

XStream Concepts

This chapter contains concepts related to XStream.

This chapter contains these topics:

See Also: Chapter 4, "Configuring XStream"

XStream Out

XStream Out can capture transactions from the redo log of an Oracle database and send them efficiently to a client application. XStream Out provides a transaction-based interface for streaming these changes to client applications. The client application can interact with other systems, including non-Oracle systems, such as non-Oracle databases or file systems.

XStream Out has both OCI and Java interfaces and supports all of the data types that are supported by Oracle Streams, including LOBs, `LONG`, `LONG RAW`, and `XMLType`.

This section contains these topics:

See Also:

- Part IV, "XStream OCI API Reference"
- *Oracle Database XStream Java API Reference*

The Outbound Server

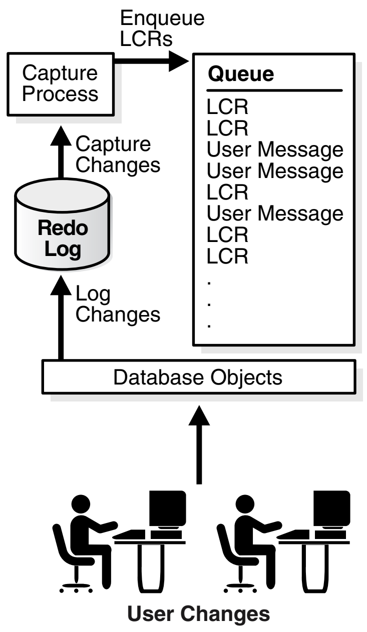
With XStream Out, an Oracle Streams apply process functions as an outbound server. An **outbound server** is an optional Oracle background process that sends database changes to a client application. Specifically, a client application can attach to an outbound server and extract database changes from LCRs. A client application attaches to the outbound server using the OCI or Java interface.

A client application can create multiple sessions. Each session can attach to only one outbound server, and each outbound server can serve only one session at a time. However, different client application sessions can connect to different outbound servers or inbound servers.

In an XStream Out configuration, a capture process captures database changes and sends these changes to an outbound server. A **capture process** is an optional Oracle background process that scans the database redo log to capture DML and DDL changes made to database objects. When a capture process is configured to capture changes from the redo log, the database where the changes were generated is called the **source database** for the capture process.

Figure 2–1 shows a capture process.

Figure 2–1 Capture Process



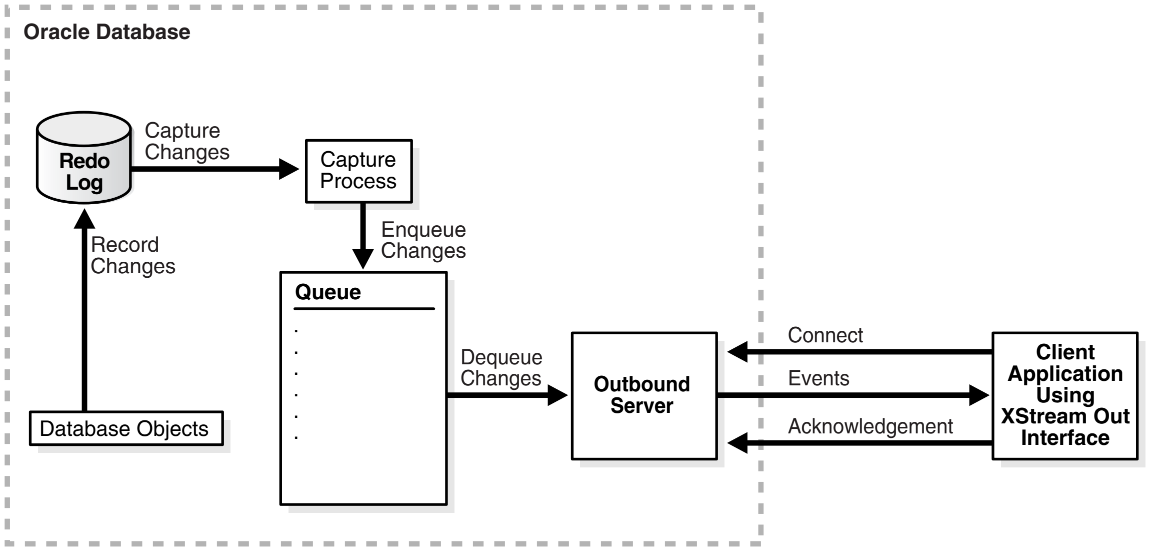
Change capture can be performed on the same database as the outbound server or on a different database. When change capture is performed on a different database from the one that contains the outbound server, a **propagation** sends the changes from the change capture database to the outbound server database. Downstream capture is also a supported mode to reduce the load on the source database.

When both the capture process and the outbound server are enabled, data changes, encapsulated in row LCRs and DDL LCRs, are sent to the outbound server. The outbound server can publish LCRs in various formats, such as OCI and Java. The client application can process LCRs that are passed to it from the outbound server or wait for LCRs from the outbound server by using a loop.

An outbound server sends LOB, `LONG`, `LONG RAW`, and `XMLType` data to the client application in chunks. Several chunks comprise a single column value of LOB, `LONG`, `LONG RAW`, or `XMLType` data type.

Figure 2–2 shows an outbound server configuration.

Figure 2–2 XStream Out Outbound Server



The client application can detach from the outbound server whenever necessary. When the client application re-attaches, the outbound server automatically determines where in the stream of LCRs the client application was when it detached. The outbound server starts sending LCRs from this point forward.

> **See Also:** *Oracle Streams Concepts and Administration* for detailed information about capture processes

Outbound Servers and Apply Process Features

An Oracle Streams apply process functions as an outbound server, but some apply process features are not applicable to an outbound server. The following sections describe which apply process features are applicable to outbound servers and which are not:

- Apply Process Features That Are Applicable to Outbound Servers
- Apply Process Features That Are Not Applicable to Outbound Servers

> **See Also:** *Oracle Streams Concepts and Administration* for information about apply processes

Apply Process Features That Are Applicable to Outbound Servers The following apply process features can be used with outbound servers:

- Rules and rule sets

  See *Oracle Streams Concepts and Administration*.

- Rule-based transformations

When a custom rule-based transformation is specified on a rule used by an outbound server, the user who calls the transformation function is the connect user for the outbound server.

See *Oracle Streams Concepts and Administration*.

- The following apply process parameters:
  - `apply_sequence_nextval`
  - `disable_on_limit`
  - `grouptransops`
  - `ignore_transaction`
  - `max_sga_size`
  - `maximum_scn`
  - `startup_seconds`
  - `time_limit`
  - `trace_level`
  - `transaction_limit`
  - `txn_age_spill_threshold`
  - `txn_lcr_spill_threshold`
  - `write_alert_log`

  These apply process parameters control the behavior of outbound servers.

  > **Note:** Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the following parameters are available: `apply_sequence_nextval`, `ignore_transaction`, `grouptransops`, and `max_sga_size`.

  See *Oracle Database PL/SQL Packages and Types Reference*.

- Transaction assembly by reader servers

  See *Oracle Streams Concepts and Administration*.

- The spilling of unapplied LCRs to hard disk

  See *Oracle Streams Concepts and Administration*.

- Instantiation system change number (SCN) settings

  Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.

  See *Oracle Streams Replication Administrator's Guide*.

Apply Process Features That Are Not Applicable to Outbound Servers The following apply process features cannot be used with outbound servers:

- Apply handlers

You cannot specify an apply handler for an outbound server. The client application can perform custom processing of the LCRs instead if necessary. However, if apply processes are configured in the same database as the outbound server, then you can specify apply handlers for these apply processes. In addition, you can configure general apply handlers for the database. An outbound server ignores general apply handlers.

See *Oracle Streams Concepts and Administration*.

- The following apply process parameters:
  - `allow_duplicate_rows`
  - `commit_serialization`
  - `compare_key_only`
  - `disable_on_error`
  - `parallelism`
  - `preserve_encryption`
  - `rtrim_on_implicit_conversion`

  Outbound servers ignore the settings for these apply process parameters.

  The `commit_serialization` parameter is always set to `FULL` for an outbound server, and the `parallelism` parameter is always set to `1` for an outbound server.

  Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `compare_key_only` parameter is available.

  See *Oracle Database PL/SQL Packages and Types Reference*.

- Apply tags

  An outbound server cannot set an apply tag for the changes it processes.

  See *Oracle Streams Replication Administrator's Guide*.

- Apply database links

  Outbound servers cannot use database links.

  See *Oracle Streams Replication Administrator's Guide*.

- Conflict detection and resolution

  An outbound server does not detect conflicts, and conflict resolution cannot be set for an outbound server.

  See *Oracle Streams Replication Administrator's Guide*.

- Dependency scheduling

  An outbound server does not evaluate dependencies because its parallelism must be 1.

  See *Oracle Streams Concepts and Administration*.

- Substitute key column settings

  An outbound server ignores substitute key column settings.

  See *Oracle Streams Concepts and Administration*.

- Enqueue directives specified by the `SET_ENQUEUE_DESTINATION` procedure in the `DBMS_APPLY_ADM` package

  An outbound server cannot enqueue changes into an Oracle database queue automatically using the `SET_ENQUEUE_DESTINATION` procedure.

  See *Oracle Database PL/SQL Packages and Types Reference*.

- Execute directives specified by the `SET_EXECUTE` procedure in the `DBMS_APPLY_ADM` package

  An outbound server ignores execute directives.

  See *Oracle Database PL/SQL Packages and Types Reference*.

- Error creation and execution

  An outbound server does not create an error transaction when it encounters an error. It records information about errors in the `ALL_APPLY` and `DBA_APPLY` views, but it does not enqueue the transaction into the error queue.

  See *Oracle Streams Concepts and Administration*.

ID Key LCRs

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

XStream Out does not support the following data types in row LCRs:

- `BFILE`
- `ROWID`
- User-defined types (including object types, REFs, varrays, and nested tables)
- XMLType stored object relationally or as binary XML
- The following Oracle supplied types: Any types, URI types, spatial types, and media types

These data type restrictions pertain to both ordinary (heap-organized) tables and index-organized tables.

ID key LCRs enable an XStream client application to process changes to rows that include unsupported data types. ID key LCRs do not contain all of the columns for a row change. Instead, they contain the rowid of the changed row, a group of key columns to identify the row in the table, and the data for the scalar columns of the table that are supported by XStream Out. ID key LCRs do not contain columns for unsupported data types.

An XStream client application can use ID key LCRs in the following ways:

- If the application does not require the data in the unsupported columns, then the application can process the values of the supported columns in the ID key LCRs normally.
- If the application requires the data in the unsupported columns, then the application can use the information in an ID key LCR to query the correct row in the database and consume the unsupported data for the row.

ID Key LCRs Demo

A demo is available that creates a sample client application that process ID key LCRs. Specifically, the client application attaches to an XStream outbound server and waits for LCRs from the outbound server. When the client application receives an ID key LCR, it can query the appropriate source database table using the rowid in the ID key LCR.

The demo is available in the following location in both OCI and Java code:

```
$ORACLE_HOME/rdbms/demo/xstream/idkey
```

Sequence LCRs

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

A **sequence LCR** is a row LCR that includes information about sequence values. Sequence database objects generate sequence values.

You can stream sequence LCRs in the following ways:

- To capture sequence LCRs using a capture process, set the capture process parameter `capture_sequence_nextval` to `Y`.
- To construct sequence LCRs using the OCI interface, use the `OCILCRNew` function and the `OCILCRHeaderSet` function with the `OCI_ROWLCR_SEQ_LCR` flag.
- To construct sequence LCRs using the Java interface, use the `DefaultRowLCR` constructor and `setSequenceLCRFlag` method.

An apply process or XStream inbound server can use sequence LCRs to ensure that the sequence values at a destination database use the appropriate values. For increasing sequences, the sequence values at the destination are equal to or greater than the sequence values at the source database. For decreasing sequences, the sequence values at the destination are less than or equal to the sequence values at the source database. To instruct an apply process or XStream inbound server to use sequence LCRs, set the `apply_sequence_nextval` apply process parameter to `Y`.

Note: Sequence LCRs are intended for one-way replication configurations. Sequence LCRs cannot be used in bi-directional replication configurations.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `capture_sequence_nextval` capture process parameter
- *Oracle Streams Concepts and Administration* for information about setting a capture process parameter
- Chapter 11, "OCI XStream Functions" for more information about the OCI interface
- *Oracle Database XStream Java API Reference* for more information about the Java interface
- *Oracle Database Administrator's Guide* for information about sequences

Considerations for XStream Outbound Servers

The following are considerations for XStream outbound servers:

- LCRs processed by an outbound server must be LCRs that were captured by a capture process. An outbound server does not support LCRs that were captured by synchronous captures or LCRs that were constructed by applications.
- A single outbound server can process captured LCRs from only one source database. The source database is the database where the changes encapsulated in the LCRs were generated in the redo log.
- The source database for the changes captured by a capture process must be at 10.2.0 or higher compatibility level for these changes to be processed by an outbound server.
- The capture process for an outbound server must be running on an Oracle Database 11*g* Release 2 (11.2) or later database.
- A single capture process cannot capture changes for both an outbound server and an apply process. However, a single capture process can capture changes for multiple outbound servers.
- An outbound server appears as an Oracle Streams apply process in Oracle Enterprise Manager.
- Automatic split and merge of a stream is possible when the capture process and the outbound server for the stream run on different database instances. However, when the capture process and outbound server for a stream run on the same database instance, automatic split and merge of the stream is not possible. See *Oracle Streams Replication Administrator's Guide* for information about automatic split and merge.

XStream Out and Distributed Transactions

You can perform distributed transactions using either of the following methods:

- Modify tables in multiple databases in a coordinated manner using database links.
- Use the XA interface, as exposed by the `DBMS_XA` supplied PL/SQL package or by the OCI or JDBC libraries. The XA interface implements X/Open Distributed Transaction Processing (DTP) architecture.

In an XStream Out configuration, changes made to the source database during a distributed transaction using either of the preceding methods are streamed to an

XStream outbound server. The outbound server sends the changes in a transaction to the XStream client application after the transaction has committed.

However, the distributed transaction state is not replicated or sent. The client application does not inherit the in-doubt or prepared state of such a transaction. Also, XStream does not replicate or send the changes using the same global transaction identifier used at the source database for XA transactions.

XA transactions can be performed in two ways:

- Tightly coupled, where different XA branches share locks
- Loosely coupled, where different XA branches do not share locks

XStream supports replication of changes made by loosely coupled XA branches regardless of the `COMPATIBLE` initialization parameter value. XStream supports replication of changes made by tightly coupled branches on an Oracle RAC source database only if the `COMPATIBLE` initialization parameter is set to `11.2.0` or higher.

See Also:

- *Oracle Database Administrator's Guide* for more information about distributed transactions
- *Oracle Database Advanced Application Developer's Guide* for more information about Oracle XA

XStream In

XStream In enables a remote client application to send information into an Oracle database from another system, such as a non-Oracle database or a file system. XStream In provides an efficient, transaction-based interface for sending information to an Oracle database from client applications. XStream In can consume the information coming into the Oracle database in several ways, including data replication, auditing, and change data capture. XStream In supports both OCI and Java interfaces.

When compared with OCI client applications that make DML changes to an Oracle database directly, XStream In is more efficient for near real-time, transaction-based, heterogeneous DML changes to Oracle databases.

XStream In uses the following features of Oracle Streams:

- High performance processing of DML changes using an apply process and, optionally, apply process parallelism
- Apply process features such as SQL generation, conflict detection and resolution, error handling, and customized processing with apply handlers
- Streaming network transmission of information with minimal network round-trips
- Rules, rule sets, and rule-based transformations

  When a custom rule-based transformation is specified on a rule used by an inbound server, the user who calls the transformation function is the apply user for the inbound server.

XStream In supports all of the data types that are supported by Oracle Streams, including LOBs, `LONG`, `LONG RAW`, and `XMLType`. A client application sends LOB and `XMLType` data to the inbound server in chunks. Several chunks comprise a single column value of LOB, `LONG`, `LONG RAW`, or `XMLType` data type.

This section contains these topics:

- The Inbound Server

- Considerations for XStream Inbound Servers

See Also:

- Part IV, "XStream OCI API Reference"
- *Oracle Database XStream Java API Reference*
- *Oracle Streams Concepts and Administration*

The Inbound Server

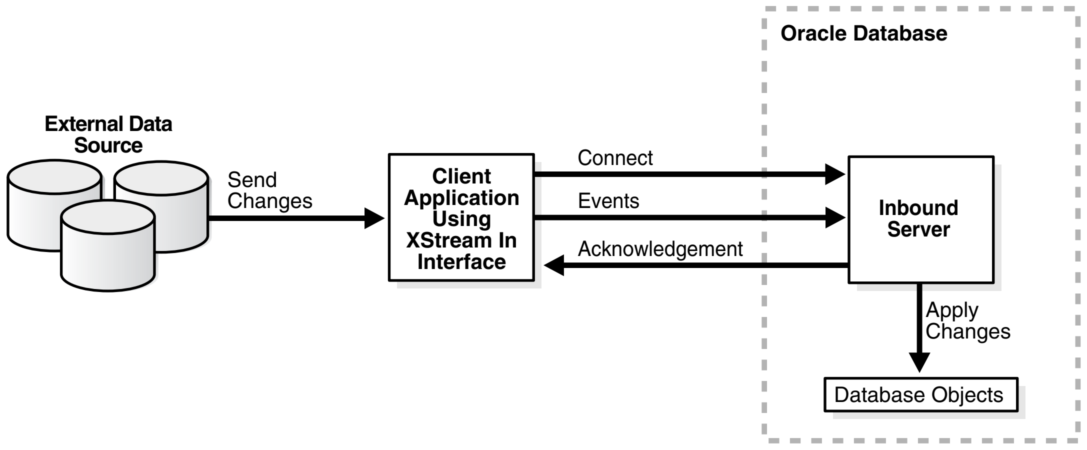
With XStream In, an Oracle Streams apply process functions as an inbound server. An **inbound server** is an optional Oracle background process that receives LCRs from a client application. Specifically, a client application can attach to an inbound server and send row changes and DDL changes encapsulated in LCRs.

An external client application connects to the inbound server using the OCI or the Java interface. After the connection is established, the client application acts as the capture agent for the inbound server by streaming LCRs to it.

A client application can create multiple sessions. Each session can attach to only one inbound server, and each inbound server can serve only one session at a time. However, different client application sessions can connect to different inbound servers or outbound servers. A client application can detach from the inbound server whenever necessary.

Figure 2–3 shows an inbound server configuration.

Figure 2–3 XStream In Inbound Server



Note: An inbound server uses a queue that is not shown in Figure 2–3. An inbound server’s queue is only used to store error transactions.

Considerations for XStream Inbound Servers

The following are considerations for XStream inbound servers:

- You can control a DML or DDL trigger's firing property using the `SET_TRIGGER_FIRING_PROPERTY` procedure in the `DBMS_DDL` package. This procedure lets you specify whether a trigger always fires, fires once, or fires for apply process changes only. When a trigger is set to fire once, it fires for changes made by a user process, but it does not fire for changes made by an apply process or inbound server. A trigger's firing property works the same for apply processes and inbound servers. See *Oracle Streams Concepts and Administration*.

- An inbound server ignores the setting for the `ignore_transaction` apply process parameter because LCRs sent to the inbound server by the client application might not have transaction ID values.

> **Note:** Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `ignore_transaction` parameter is available for outbound servers and apply processes.

- An inbound server ignores the setting for the `maximum_scn` apply process parameter because LCRs sent to the inbound server by the client application might not have SCN values.
- Currently, an inbound server appears as an Oracle Streams apply process in Oracle Enterprise Manager.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about apply process parameters

Position Order in an LCR Stream

The following sections describe the position order in an LCR stream for both XStream Out and XStream In:

- About Position Order
- Position of LCRs and XStream Out
- Position of LCRs and XStream In
- Summary of Position Use in XStream Out and XStream In

About Position Order

Both XStream Out and XStream In use LCR streams to share transactions. XStream Out sends LCR streams to a client application. XStream In receives LCR streams from a client application.

Each LCR has a position attribute. The **position** of an LCR identifies its placement in the stream of LCRs in a transaction. Each LCR position has the following properties:

- The position is unique for each LCR.
- The position is of `RAW` data type.
- The position is strictly increasing within the LCR stream, within a transaction, and across transactions.
- The position is byte-comparable, and the comparison results for multiple positions determines the ordering of the LCRs in the stream.
- The position of an LCR remains identical when the database, the client application, or an XStream component restarts.
- The position is not affected by any rule changes that might reduce or increase the number of LCRs in the stream.

XStream Out only sends committed data, and XStream In only receives committed data.

The following are the properties related to an LCR stream:

- An LCR stream must be repeatable.
- An LCR stream must contain a list of assembled, committed transactions. LCRs from one transaction are contiguous. There is no interleaving of transactions in an LCR stream.
- Each transaction within an LCR stream must have an ordered list of LCRs and a transaction ID.
- The last LCR in each transaction must be a commit LCR.
- Each LCR must have a unique position.
- The position of all LCRs within a single transaction and across transactions must be strictly increasing.

An LCR stream can batch LCRs from multiple transactions and arrange them in increasing position order. LCRs from one transaction are contiguous, and the position must be increasing in the transaction. Also, the position must be nonzero for all LCRs.

Position of LCRs and XStream Out

An XStream Out outbound server streams LCRs that were captured by a capture process to a client application. This section describes concepts related to the LCR positions for an outbound server.

Additional LCR Attributes Related to Position

LCRs that were captured by a capture process contain the following additional attributes related to LCR position:

- The `scn_from_position` attribute contains the SCN of the LCR.
- The `commit_scn_from_position` attribute contains the commit SCN of the transaction to which the LCR belongs.

Note: The `scn_from_position` and `commit_scn_from_position` attributes are not present in row LCRs captured by a synchronous capture nor in explicitly captured row LCRs.

See Also: *Oracle Database PL/SQL Packages and Types Reference*

The Processed Low Position and Restartability for XStream Out

If the outbound server or the client application stops abnormally, then the connection between the two is broken automatically. In this case, the client application must roll back all incomplete transactions.

The **processed low position** is a position below which all transactions have been processed by the client application. The client application must maintain its processed low position to recover properly after either it or the outbound server (or both) are restarted. The processed low position indicates that the client application has processed all LCRs that are less than or equal to this value. The client application can update the processed low position for each transaction that it consumes.

When the client application attaches to the outbound server, the following conditions related to the processed low position are possible:

- The client application can pass a processed low position to the outbound server that is equal to or greater than the outbound server’s processed low position. In this case, the outbound server resumes streaming LCRs from the first LCR that has a position greater than the client application’s processed low position.
- The client application can pass a processed low position to the outbound server that is less than the outbound server’s processed low position. In this case, the outbound server raises an error.
- The client application can pass `NULL` to the outbound server. In this case, the outbound server determines the processed low position automatically and starts streaming LCRs from the LCR that has a position greater than this processed low position. When this happens, the client application must suppress or discard each LCR with a position less than or equal to the client application’s processed low position.

See Also: "Displaying the Processed Low Position for an Outbound Server" on page 6-8

Streaming Network Transmission

To minimize network latency, the outbound server streams LCRs to the client application with time-based acknowledgments. For example, the outbound server might send an acknowledgment every 30 seconds. This streaming protocol fully utilizes the available network bandwidth, and the performance is unaffected by the presence of a wide area network (WAN) separating the sender and the receiver. The outbound server extends the underlying Oracle Streams infrastructure, and the outbound server maintains the streaming performance rate.

Using OCI, you can control the time period of the interval by setting the `OCI_ATTR_XSTREAM_ACK_INTERVAL` attribute through the OCI client application. The default is 30 seconds.

Using Java, you can control the time period of the interval by setting the `batchInterval` parameter in the `attach` method in the `XStreamOut` class. The client application can specify this interval when it invokes the `attach` method.

If the interval is large, then the outbound server can stream out more LCRs for each acknowledgment interval. However, a longer interval delays how often the client application can send the processed low position to the outbound server. Therefore, a longer interval might mean that the processed low position maintained by the outbound server is not current. In this case, when the outbound server restarts, it must start processing LCRs at an earlier position than the one that corresponds to the processed low position maintained by the client application. Therefore, more LCRs might be retransmitted, and the client application must discard the ones that have been applied.

Position of LCRs and XStream In

A client application streams LCRs to an XStream In inbound server. This section describes concepts related to the LCR positions for an inbound server.

Each position must be encoded in a format (such as base-16 encoding) that supports byte comparison. The position is essential to the total order of the transaction stream sent by client applications using the XStream In interface.

The following positions are important for inbound servers:

- The **applied low position** indicates that the LCRs less than or equal to this value have been applied.

  An LCR is applied by an inbound server when the LCR has either been executed, sent to an apply handler, or moved to the error queue.
- The **spill position** indicates that the LCRs with positions less than or equal to this value have either been applied or spilled from memory to hard disk.
- The **applied high position** indicates the highest position of an LCR that has been applied.

  When the `commit_serialization` apply process parameter is set to `DEPENDENT_TRANSACTIONS` for an inbound server, an LCR with a higher commit position might be applied before an LCR with a lower commit position. When this happens, the applied high position is different from the applied low position.
- The processed low position is the higher value of either the applied low position or the spill position.

  The processed low position is the position below which the inbound server no longer requires any LCRs. This position corresponds with the oldest SCN for an Oracle Streams apply process that applies changes captured by a capture process.

  The processed low position indicates that the LCRs with positions less than or equal to this position have been processed by the inbound server. If the client re-attaches to the inbound server, then it must send only LCRs with positions greater than the processed low position because the inbound server discards any LCRs with positions less than or equal to the processed low position.

If the client application stops abnormally, then the connection between the client application and the inbound server is automatically broken. Upon restart, the client application retrieves the processed low position from the inbound server and instructs its capture agent to retrieve changes starting from this processed low position.

To limit the recovery time of a client application using the XStream In interface, the client application can send activity, such as empty transactions, periodically to the inbound server. Row LCRs can include commit transaction control directives. When there are no LCRs to send to the server, the client application can send a row LCR with a commit directive to advance the inbound server's processed low position. This activity acts as an acknowledgment so that the inbound server's processed low position is advanced.

Example 2–1 Advancing the Processed Low Position of an Inbound Server

Consider a client application and an external data source. The client application sends changes made to the `hr.employees` table to the inbound server for processing, but the external data source includes many other tables, including the `oe.orders` table.

Assume that the following changes are made to the external data source:

Position	Change	Client Application Activity
1	Insert into the `hr.employees` table	Send row LCR including the change to the inbound server
2	Insert into the `oe.orders` table	None
3	Commit	Send a row LCR with a commit directive to inbound server
4	Insert into the `oe.orders` table	None

Position	Change	Client Application Activity
5	Update the `oe.orders` table	None
6	Commit	None
7	Commit	None
...	... (Activity on the external data source, but no changes to the `hr.employees` table)	None
100	Insert into the `oe.orders` table	None
101	Commit	None

The client application gets the changes from the external data source, generates appropriate LCRs, and sends the LCRs to the inbound server. Therefore, the inbound server receives the following LCRs:

- Row LCR for position 1
- Row LCR for position 3

After position 3, there are no relevant changes to send to the inbound server. If the inbound server restarts when the client application has processed all the changes up to position 101, then, after restarting, the client application must recheck all of the external database changes from position 4 forward. The rechecks are required because the inbound server's processed low position is 3.

Instead, assume that the client application sends commits to the inbound server periodically, even when there are no relevant changes to the `hr.employees` table:

Position	Change	Client Application Activity
1	Insert into the `hr.employees` table	Send row LCR including the change to the inbound server
2	Insert into the `oe.orders` table	None
3	Commit	Send a row LCR with a commit directive to inbound server
4	Insert into the `oe.orders` table	None
5	Update the `oe.orders` table	None
6	Commit	None
7	Commit	None
...	... (Activity on the external data source, but no changes to the `hr.employees` table)	Send several row LCRs, each one with a commit directive, to the inbound server
100	Insert into the `oe.orders` table	None
101	Commit	Send a row LCR with a commit directive to the inbound server

In this case, the inbound server moves its processed low position to 101 when it has processed all of the row LCRs sent by the client application. If the inbound server restarts, its processed low position is 101, and the client application does not need to check all of the changes back to position 3.

The sample applications in "Sample XStream Client Application" on page 4-26 include code that sends a row LCR with a commit directive to an inbound server. These commit directives are sometimes called "ping LCRs." Search for the word "ping" in the sample XStream client applications to find the parts of the applications that include this code.

See Also: "Displaying the Position Information for an Inbound Server" on page 6-11

Summary of Position Use in XStream Out and XStream In

Table 2–1 compares how an XStream Out outbound server and an XStream In inbound server use positions.

Table 2–1 Position Use in the Outbound Server and the Inbound Server

XStream Out Outbound Server	XStream In Inbound Server
The outbound server exposes the position.	The client application sets the position.
If the outbound server or client application stops abnormally, then all LCRs above the processed low position are resent. The processed low position is equivalent to an apply process low watermark (LWM), and the apply process obtains the oldest SCN value by using this value.	If the inbound server or client application stops abnormally, then the client application must retransmit all LCRs with a position greater than or equal to the processed low position. The processed low position is equivalent to the apply process low water mark (LWM).

XStream and SQL Generation

SQL generation is the ability to generate the SQL statement required to perform the change encapsulated in a row LCR. Apply processes, XStream outbound servers, and XStream inbound servers can use SQL generation to generate the SQL statement necessary to perform the insert, update, or delete operation in a row LCR.

This section contains these topics:

- Interfaces for Performing SQL Generation
- SQL Generation Formats
- Data Types and Character Sets
- Interfaces for Performing SQL Generation

See Also: *Oracle Streams Concepts and Administration*

Interfaces for Performing SQL Generation

You can use the following interfaces to perform SQL generation:

- The PL/SQL interface, which uses the `GET_ROW_TEXT` and `GET_WHERE_CLAUSE` member procedures for row LCRs
- The OCI for XStream
- The Java interface for XStream

The PL/SQL interface generates SQL in a `CLOB` data type, while the OCI and Java interfaces generate SQL in plain text. In the Java interface, the size of the text is limited by the size of `String` data type.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `GET_ROW_TEXT` and `GET_WHERE_CLAUSE` row LCR member procedures
- Part IV, "XStream OCI API Reference"
- *Oracle Database XStream Java API Reference* for information about the Java interface for XStream

SQL Generation Formats

SQL statements can be generated in one of two formats: inline values or bind variables. Use inline values when the returned SQL statement is relatively small. For larger SQL statements, use bind variables. In this case, the bind variables are passed to the client application in a separate list that includes pointers to both old and new column values.

For information about using bind variables with each interface, refer to the following documentation:

- The documentation for the `GET_ROW_TEXT` and `GET_WHERE_CLAUSE` row LCR member procedures in *Oracle Database PL/SQL Packages and Types Reference*
- "OCILCRRowStmtWithBindVarGet()" on page 11-16
- The documentation for `DefaultRowLCR.getBinds()` in *Oracle Database XStream Java API Reference*

Note: For generated SQL statements with the values inline, SQL injection is possible. SQL injection is a technique for maliciously exploiting applications that use client-supplied data in SQL statements, thereby gaining unauthorized access to a database to view or manipulate restricted data. Oracle strongly recommends using bind variables if you plan to execute the generated SQL statement. See *Oracle Database PL/SQL Language Reference* for more information about SQL injection.

Data Types and Character Sets

Regarding data types and character sets, SQL generation works the same way for XStream Out outbound servers, XStream In inbound servers, and apply processes. For detailed information, see *Oracle Streams Concepts and Administration*.

SQL Generation Demo

A demo that performs SQL generation is available. The demo uses the `DBMS_XSTREAM_ADM` PL/SQL package to configure an XStream Out environment, and it uses an OCI client application to perform SQL generation.

The demo uses SQL generation to replicate DML changes from a source database to a destination database. Specifically, the demo creates the `xsdemosg` schema in both the source database and the destination database. It creates various types of tables in the `xsdemosg` schema at each database, including tables with LOB columns. It executes a set of DML statements on the tables in `xsdemosg` schema in the source database. Oracle Streams components, such as a capture process and a queue, send the changes in the form of LCRs to an XStream outbound server that is also running on the source database. The outbound server makes the LCRs available to the demo client application.

The demo client application, when run, uses the OCI API to connect to the outbound server and receive the LCRs. The demo client application uses SQL generation to execute the changes that are encapsulated in the LCRs. Therefore, the client application replicates the changes made to `xsdemosg` schema in the source database to the `xsdemosg` in the destination database.

You can modify the demo to replicate changes to any schema. Both the schema and the replicated tables must exist on both the source database and the destination database. SQL generation is only possible for DML changes. Therefore, this demo cannot be used to replicate DDL changes.

This demo is available in the following location:

```
$ORACLE_HOME/rdbms/demo/xstream/sqlgen
```

Note: The SQL generation demo is not available for the XStream Java API.

XStream and Security

XStream Out allows a user to receive LCRs. After an XStream Out user receives LCRs, the user might save the contents of LCRs to a file or generate the SQL statements to execute the LCRs on a non-Oracle database. XStream In allows a user to update tables in its own schema. XStream does not assume that the connected user to the outbound server or inbound server is trusted.

Java and OCI client applications must connect to an Oracle database before attaching to an XStream outbound server created on that database. The connected user must be the same as the connect user configured for the outbound server. Otherwise, an error is raised.

Java and OCI client applications must connect to an Oracle database before attaching to an XStream inbound server created on that database. The connected user must be the same as the apply user configured for the inbound server. Otherwise, an error is raised.

The XStream Java layer API relies on Oracle JDBC security because XStream accepts the Oracle JDBC connection instance created by client applications in the XStream `attach` API. The connected user is validated as an XStream user.

See Also:

- "Security Model" on page 8-4 for information about the security requirements for configuring and managing XStream
- Chapter 4, "Configuring XStream"
- *Oracle Streams Concepts and Administration* for information about apply users

Other Ways to Share Information in a Heterogeneous Environment

Oracle Streams provides other ways to implement heterogeneous information sharing besides XStream, both in past releases and in the current release. These ways include:

- Replicating data changes to a non-Oracle database using an Oracle Database Gateway
- Dequeuing messages from an Oracle database using a Java Message Service (JMS) client
- Enqueuing messages directly into an Oracle database queue with a client application

See Also:

- *Oracle Streams Replication Administrator's Guide*
- *Oracle Database 2 Day + Data Replication and Integration Guide*
- *Oracle Streams Advanced Queuing User's Guide*

3 XStream Use Cases

XStream provides a flexible infrastructure for sharing information between Oracle data sources and non-Oracle data sources. Therefore, you can use XStream in many different ways to meet the needs of various organizations. This chapter describes the most common use cases for XStream.

This chapter contains these topics:

See Also:

- Chapter 2, "XStream Concepts"
- Chapter 4, "Configuring XStream"
- Chapter 5, "Managing XStream"
- Chapter 6, "Monitoring XStream"
- Chapter 7, "Troubleshooting XStream"

Introduction to XStream Use Cases

Each XStream use case in this chapter contains three main elements:

- A general description of the use case as it applies to both XStream Out and XStream In
- A specific scenario for XStream Out
- A specific scenario for XStream In

In some cases, a section includes a reference to sample code in the Oracle Database installation that illustrates a scenario.

XStream Out Use Cases

In each XStream Out use case, the following components and actions send Oracle Database changes to a client application:

- Oracle Streams captures data changes made to an Oracle database.
- Oracle Streams sends these changes, in the form of logical change records (LCRs), to an outbound server.

- The outbound server sends the LCRs to a client application.

How the client application processes the LCRs is different for each use case.

See Also: "XStream Out" on page 2-1

XStream In Use Cases

In each XStream In use case, the following components and actions send Oracle Database changes to an inbound server:

- A client application gathers data changes from an external data source and sends them to an inbound server in the form of LCRs.
- The inbound server receives the LCRs from a client application.
- The inbound server can apply the data changes to database objects in an Oracle database. The inbound server can also process the LCRs in a customized way.

How the client application gathers the data changes is different for each use case.

See Also: "XStream In" on page 2-9

Replicating Data Changes with Non-Oracle Databases

You can configure a heterogeneous replication environment with XStream. Replication is generally used to improve availability and to improve performance by spreading the network load over multiple regions and servers. In a heterogeneous replication environment, data is replicated between databases from different vendors. See *Oracle Streams Replication Administrator's Guide* for common reasons to use replication.

XStream Out can send data changes made to an Oracle database to a non-Oracle database. Specifically, the client application connects to the outbound server and receives changes made to tables within the Oracle database. The client application then applies the data changes in the LCRs to the non-Oracle database. The client application can process the LCRs in any customized way before applying them.

XStream In can receive data changes made to a non-Oracle database. Specifically, the client application gathers the data changes made to the non-Oracle database, formats these changes into LCRs, and sends these LCRs to an inbound server. The inbound server applies the changes in the LCRs to the Oracle database.

Using Files to Store Data Changes

Some environments use files to store data changes. Typically, files store data changes for the following reasons:

- To process data changes in an environment that has no physical network or a limited physical network. For example, some locations do not have a physical network for security reasons.
- To process data changes in an environment that uses disconnected computing. For example, a salesperson might fill orders on a laptop at various locations without a network connection, and then update a primary database over the network once a day.
- To process data changes in an environment that uses satellite communications. In this case, a bulk transfer of files is more efficient than incremental changes over the network.

XStream Out can send Oracle Database changes to a file in a file system. Specifically, the client application writes the data changes in LCRs to the file. The client application can process the LCRs in any customized way before writing them to the file, and the file can reside on the computer system running the client application or on a different computer system. Using SQL generation, the client application can also write the SQL statement required to perform the change encapsulated in a row LCR to a file.

XStream In can send data changes from a file to an Oracle database. Specifically, the client application reads the data changes from the file and sends the changes, in the form of LCRs, to an inbound server.

See Also:

- "XStream and SQL Generation" on page 2-16
- *Oracle Streams Concepts and Administration*

XStream Demo That Replicates Database Changes Using Files

A demo is available that creates sample client applications that perform file-based replication using the XStream APIs. Specifically, at one database, the demo creates an XStream Out configuration that captures database changes and sends the LCRs to an outbound server. A client application attaches to the outbound server and writes the database changes to a file.

At a different database, the demo creates an XStream In client application that attaches to an inbound server, reads the changes in the file, and sends them in the form of LCRs to the inbound server. The inbound server applies the changes to the database objects at the destination database.

This demo is available in the following location:

```
$ORACLE_HOME/rdbms/demo/xstream/fbr
```

Sharing Data Changes with a Client-Side Memory Cache

Some environments cache data in memory to improve performance. Cached data can provide low response times and high throughput for systems that require the best possible performance. XStream can share data changes incrementally with a client side memory cache.

XStream Out can incrementally refresh a client-side memory cache by sending Oracle database changes to a memory cache. Specifically, the client application applies the data changes in the LCRs to the memory cache. The client application can process the LCRs in any customized way before applying them, and the memory cache can reside on the computer system running the client application or on a different computer system.

XStream In can incrementally retrieve data changes from a memory cache. Specifically, the client application retrieves the data changes and sends the changes, in the form of LCRs, to an inbound server. The memory cache can reside on the computer system running the client application or on a different computer system.

Part II

XStream Administration

This part describes XStream administration. This part contains the following chapters:

4 Configuring XStream

This chapter describes configuring the Oracle Database components that are used by XStream. This chapter also includes sample client applications that communicate with an XStream outbound server and inbound server.

This chapter contains these topics:

- Preparing for XStream
- Configuring XStream Out
- Configuring XStream In
- Sample XStream Client Application

See Also:

- Chapter 2, "XStream Concepts"
- Chapter 3, "XStream Use Cases"
- Chapter 6, "Monitoring XStream"
- Chapter 7, "Troubleshooting XStream"
- Part IV, "XStream OCI API Reference"
- *Oracle Database XStream Java API Reference*

Preparing for XStream

This section describes preparing for an XStream configuration.

This section contains the following topics:

- Granting Privileges for the XStream Administrator
- Preparing for XStream Out
- Preparing for XStream In

Granting Privileges for the XStream Administrator

An XStream administrator configures and manages XStream components in an XStream Out or XStream In environment. This section describes configuring an XStream administrator by granting a user the appropriate privileges. You must configure an XStream administrator in each Oracle database included in the XStream configuration.

Prerequisites

Before configuring an XStream administrator, ensure that the following prerequisites are met:

- Ensure that you can log in to each database in the XStream configuration as an administrative user who can create users, grant privileges, and create tablespaces.
- Identify a user who will be the XStream administrator. Either create a new user with the appropriate privileges or grant these privileges to an existing user.

  Do not use the `SYS` or `SYSTEM` user as an XStream administrator, and ensure that the XStream administrator does not use the `SYSTEM` tablespace as its default tablespace.
- If a new tablespace is required for the XStream administrator, then ensure that there is enough disk space on each computer system in the XStream configuration for the tablespace. The recommended size of the tablespace is 25 MB.

Assumptions

This section makes the following assumptions:

- The username of the XStream administrator is `xstrmadmin`.
- The tablespace used by the XStream administrator is `xstream_tbs`.

To configure an XStream administrator:

1. In SQL*Plus, connect as an administrative user who can create users, grant privileges, and create tablespaces. Remain connected as this administrative user for all subsequent steps.

   See Also: *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus

2. Either create a tablespace for the XStream administrator or use an existing tablespace.

   This tablespace stores any objects created in the XStream administrator's schema, including any spillover of messages from the buffered queues owned by the schema.

   For example, the following statement creates a new tablespace for the XStream administrator:

   ```
   CREATE TABLESPACE xstream_tbs DATAFILE '/usr/oracle/dbs/xstream_tbs.dbf'
     SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
   ```

3. Create a new user to act as the XStream administrator or identify an existing user.

   For example, to create a user named `xstrmadmin` and specify that this user uses the `xstream_tbs` tablespace, run the following statement:

   ```
   CREATE USER xstrmadmin IDENTIFIED BY password
     DEFAULT TABLESPACE xstream_tbs
     QUOTA UNLIMITED ON xstream_tbs;
   ```

   Note: Enter an appropriate password for the administrative user.

See Also: *Oracle Database Security Guide* for guidelines about choosing passwords

4. Grant the Oracle Streams administrator the DBA role:

   ```
   GRANT DBA TO xstrmadmin;
   ```

   Note: The DBA role is required for a user to create or alter outbound servers, inbound servers, capture processes, synchronous captures, and apply processes. When the user does not need to perform these tasks, the DBA role can be revoked from the user.

5. Run the GRANT_ADMIN_PRIVILEGE procedure in the DBMS_XSTREAM_AUTH package.

   A user must have explicit EXECUTE privilege on a package to execute a subprogram in the package inside of a user-created subprogram, and a user must have explicit SELECT privilege on a data dictionary view to query the view inside of a user-created subprogram. These privileges cannot be granted through a role. You can run the GRANT_ADMIN_PRIVILEGE procedure to grant such privileges to the XStream administrator, or you can grant them directly.

   Depending on the parameter settings for the GRANT_ADMIN_PRIVILEGE procedure, it either grants the privileges for an XStream administrator directly, or it generates a script that you can edit and then run to grant these privileges.

   Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the DBMS_XSTREAM_AUTH package is available.

   See Also: "GRANT_ADMIN_PRIVILEGE Procedure" on page 9-6

   Use the GRANT_ADMIN_PRIVILEGE procedure to grant privileges directly:

   Run the following procedure:

   ```
   BEGIN
      DBMS_XSTREAM_AUTH.GRANT_ADMIN_PRIVILEGE(
         grantee          => 'xstrmadmin',
         grant_privileges => TRUE);
   END;
   /
   ```

   Use the GRANT_ADMIN_PRIVILEGE procedure to generate a script:

   Complete the following steps:

   a. Use the SQL statement CREATE DIRECTORY to create a directory object for the directory into which you want to generate the script. A directory object is similar to an alias for the directory. For example, to create a directory object called xstrm_dir for the /usr/admin directory on your computer system, run the following procedure:

      ```
      CREATE DIRECTORY xstrm_dir AS '/usr/admin';
      ```

   b. Run the GRANT_ADMIN_PRIVILEGE procedure to generate a script named grant_xstrm_privs.sql and place this script in the /usr/admin directory on your computer system:

```
BEGIN
   DBMS_XSTREAM_AUTH.GRANT_ADMIN_PRIVILEGE(
      grantee          => 'xstrmadmin',
      grant_privileges => FALSE,
      file_name        => 'grant_xstrm_privs.sql',
      directory_name   => 'xstrm_dir');
END;
/
```

Notice that the `grant_privileges` parameter is set to `FALSE` so that the procedure does not grant the privileges directly. Also, notice that the directory object created in Step a is specified for the `directory_name` parameter.

c. Edit the generated script if necessary and save your changes.

d. Execute the script in SQL*Plus:

```
SET ECHO ON
SPOOL grant_xstrm_privs.out
@/usr/admin/grant_xstrm_privs.sql
SPOOL OFF
```

e. Check the spool file to ensure that all of the grants executed successfully. If there are errors, then edit the script to correct the errors and rerun it.

6. If necessary, grant the following additional privileges:

   - If you plan to use Oracle Enterprise Manager to manage databases with XStream components, then configure the XStream administrator to be a Database Control administrator. Doing so grants additional privileges required by Oracle Enterprise Manager, such as the privileges required to run Oracle Enterprise Manager jobs. See *Oracle Database 2 Day DBA* for instructions.
   - Grant the privileges for a remote XStream administrator to perform actions in the local database. Grant these privileges using the `GRANT_REMOTE_ADMIN_ACCESS` procedure in the `DBMS_XSTREAM_AUTH` package. Grant this privilege if a remote XStream administrator will use a database link that connects to the local XStream administrator to perform administrative actions. Specifically, grant these privileges if either of the following conditions are true:
     – You plan to configure a downstream capture process at a remote downstream database that captures changes originating at the local source database, and the downstream capture process will use a database link to perform administrative actions at the source database.
     – You plan to use a remote XStream administrator to set the instantiation system change number (SCN) values for replicated database objects at the local database.
   - If no apply user is specified for an inbound server, then grant the XStream administrator the necessary privileges to perform DML and DDL changes on the apply objects owned by other users. If an apply user is specified, then the apply user must have these privileges. These privileges can be granted directly or through a role.
   - If no apply user is specified for an inbound server, then grant the XStream administrator `EXECUTE` privilege on any PL/SQL subprogram owned by another user that is executed by an inbound server. These subprograms can be used in apply handlers or error handlers. If an apply user is specified, then the

apply user must have these privileges. These privileges must be granted directly. They cannot be granted through a role.

- Grant the XStream administrator EXECUTE privilege on any PL/SQL function owned by another user that is specified in a custom rule-based transformation for a rule used by a capture process, synchronous capture, propagation, outbound server, or inbound server. For a capture process or synchronous capture, if a capture user is specified, then the capture user must have these privileges. For an inbound server, if an apply user is specified, then the apply user must have these privileges. These privileges must be granted directly. They cannot be granted through a role.
- Grant the XStream administrator privileges to alter database objects where appropriate. For example, if the XStream administrator must create a supplemental log group for a table in another schema, then the XStream administrator must have the necessary privileges to alter the table. These privileges can be granted directly or through a role.
- If the XStream administrator does not own the queue used by a capture process, synchronous capture, propagation, outbound server, or inbound server, and is not specified as the queue user for the queue when the queue is created, then the XStream administrator must be configured as a secure queue user of the queue if you want the XStream administrator to be able to enqueue messages into or dequeue messages from the queue. The XStream administrator might also need ENQUEUE or DEQUEUE privileges on the queue, or both. See *Oracle Streams Concepts and Administration* for information about managing queues.
- Grant the XStream administrator EXECUTE privilege on any object types that the XStream administrator might need to access. These privileges can be granted directly or through a role.
- If the XStream administrator will use Data Pump to perform export and import operations on database objects in other schemas during an instantiation, then grant the EXP_FULL_DATABASE and IMP_FULL_DATABASE roles to the XStream administrator.
- If you are using Oracle Database Vault, then the following additional privileges are required:
  - The XStream administrator must be granted the DV_STREAMS_ADMIN role to perform the following tasks: create a capture process, create an outbound server, and modify the capture user for a capture process. When the XStream administrator is not performing these tasks, you can revoke DV_STREAMS_ADMIN role from the XStream administrator.
  - The apply user for an inbound server must be authorized to apply changes to realms that include replicated database objects. The replicated database objects are the objects to which the inbound server applies changes.

    To authorize an apply user for a realm, run the DBMS_MACADM.ADD_AUTH_TO_REALM procedure and specify the realm and the apply user. For example, to authorize apply user xstrmadmin for the sales realm, run the following procedure:

```
BEGIN
 DBMS_MACADM.ADD_AUTH_TO_REALM(
  realm_name  => 'sales',
  grantee     => 'xstrmadmin');
END;
```

```
/
```

In addition, the user who performs the following actions must be granted the `BECOME USER` system privilege:

– Creates or alters a capture process

– Creates or alters an outbound server

– Creates or alters an inbound server

Granting the `BECOME USER` system privilege to the user who performs these actions is not required if Oracle Database Vault is not installed. You can revoke the `BECOME USER` system privilege from the user after the completing one of these actions, if necessary.

See *Oracle Database Vault Administrator's Guide*.

7. Repeat all of the previous steps at each Oracle database in the environment that will use XStream.

Preparing for XStream Out

This section describes the decisions to make and the tasks to complete to prepare for an XStream Out configuration.

- Decide How to Configure XStream
- Prerequisites for Configuring XStream Out

Decide How to Configure XStream

When you configure XStream Out, you must configure XStream components to capture database changes and send these changes to the outbound server in the form of logical change records (LCRs). These components include a capture process and at least one queue. The capture process can be a local capture process or a downstream capture process. For some configurations, you must also configure a propagation.

Local capture means that a capture process runs on the source database. Downstream capture means that a capture process runs on a database other than the source database. The source database is the database where the changes were generated. The primary reason to use downstream capture is to reduce the load on the source database, thereby improving its performance. The primary reason to use a local capture is because it is easier to configure and maintain.

The database that captures changes made to the source database is called the **capture database**. One of the following databases can be the capture database:

- Source database (local capture)
- Destination database (downstream capture)
- A third database (downstream capture)

If the database running the outbound server is not the capture database, then a propagation sends changes from the capture database to the database running the outbound server. If the database running the outbound server is the capture database, then this propagation between databases is not needed because the capture process and outbound server use the same queue.

You can configure the components in the following ways:

- **Local capture and outbound server in the same database:** The database objects, capture process, and outbound server are all in the same database. This configuration is the easiest to configure and maintain because all of the components are contained in one database. See Figure 4–1 for an overview of this configuration.

- **Local capture and outbound server in different databases:** The database objects and capture process are in one database, and the outbound server is in another database. A propagation sends LCRs from the source database to the outbound server database. This configuration is best when you want easy configuration and maintenance and when you want to optimize the performance of the outbound server database. See Figure 4–2 for an overview of this configuration.
- **Downstream capture and outbound server in the same database:** The database objects are in one database, and the capture process and outbound server are in another database. This configuration is best when you want to optimize the performance of the database with the database objects and want to offload change capture to another database. With this configuration, most of the components run on the database with the outbound server. See Figure 4–3 for an overview of this configuration.
- **Downstream capture and outbound server in different databases:** The database objects are in one database, the outbound server is in another database, and the capture process is in a third database. This configuration is best when you want to optimize the performance of both the database with the database objects and the database with the outbound server. With this configuration, the capture process runs on a third database, and a propagation sends LCRs from the capture database to the outbound server database. See Figure 4–4 for an overview of this configuration.

The following figures illustrate these different configurations.

Figure 4–1 Local Capture and Outbound Server in the Same Database

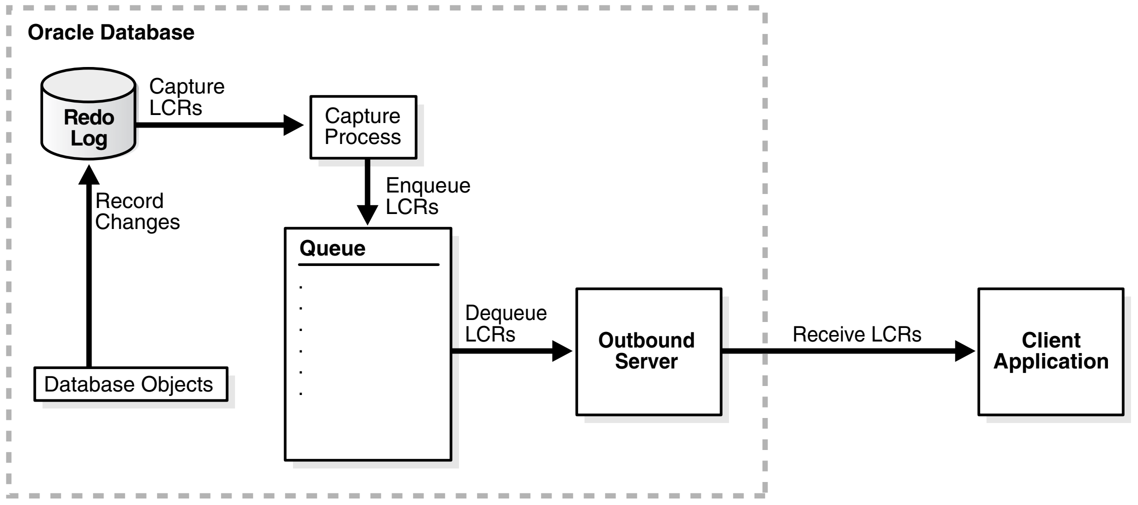


Figure 4–2 Local Capture and Outbound Server in Different Databases

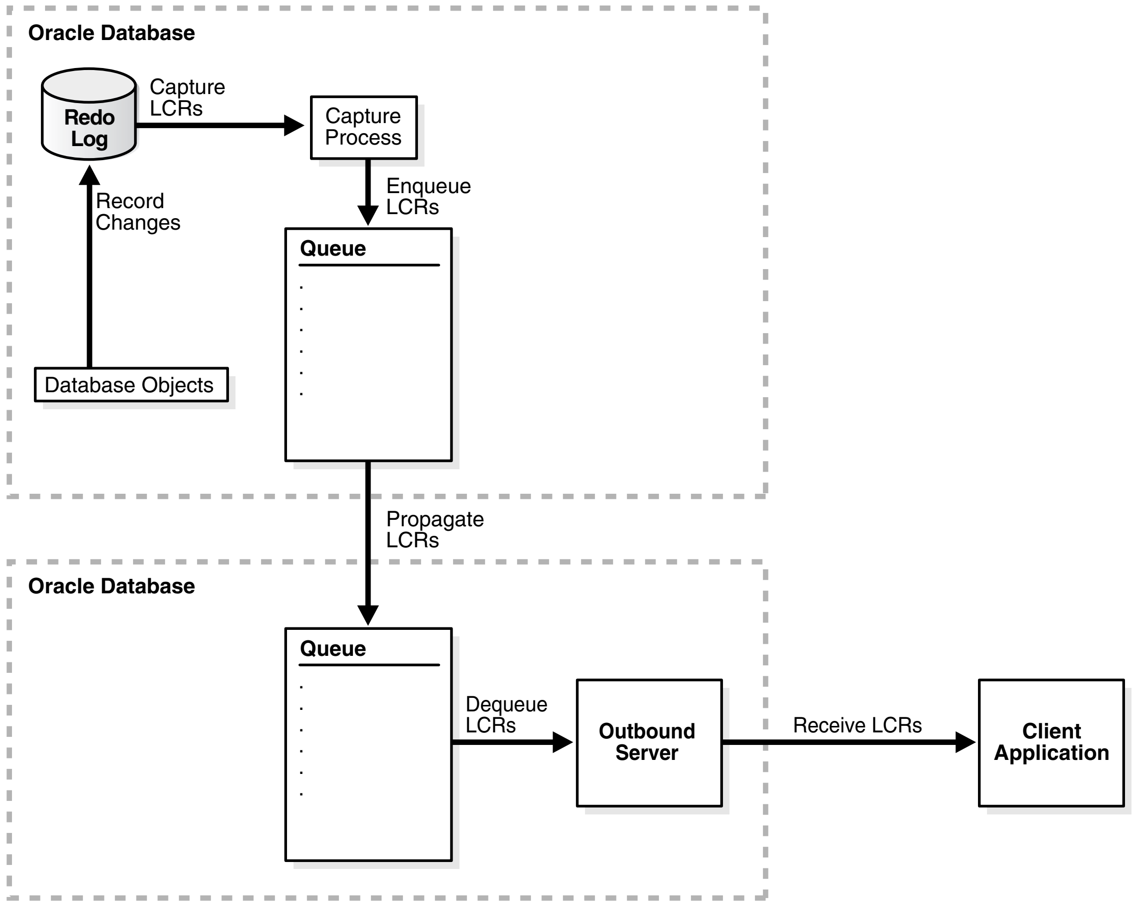


Figure 4–3 Downstream Capture and Outbound Server in the Same Database

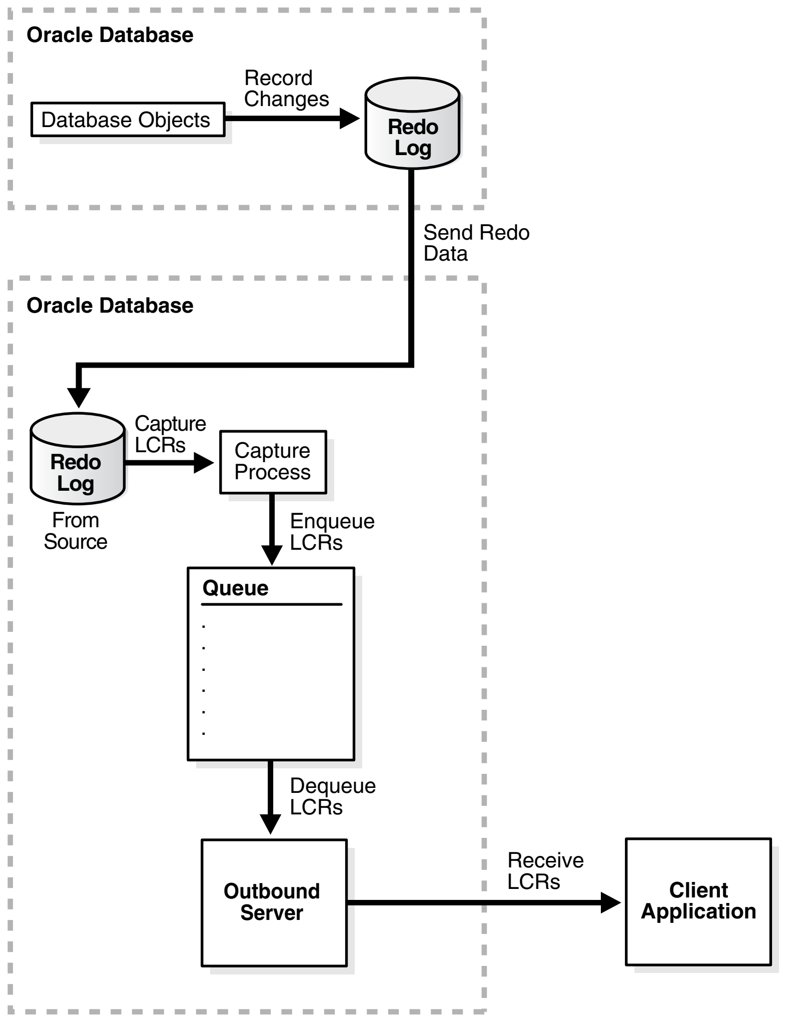
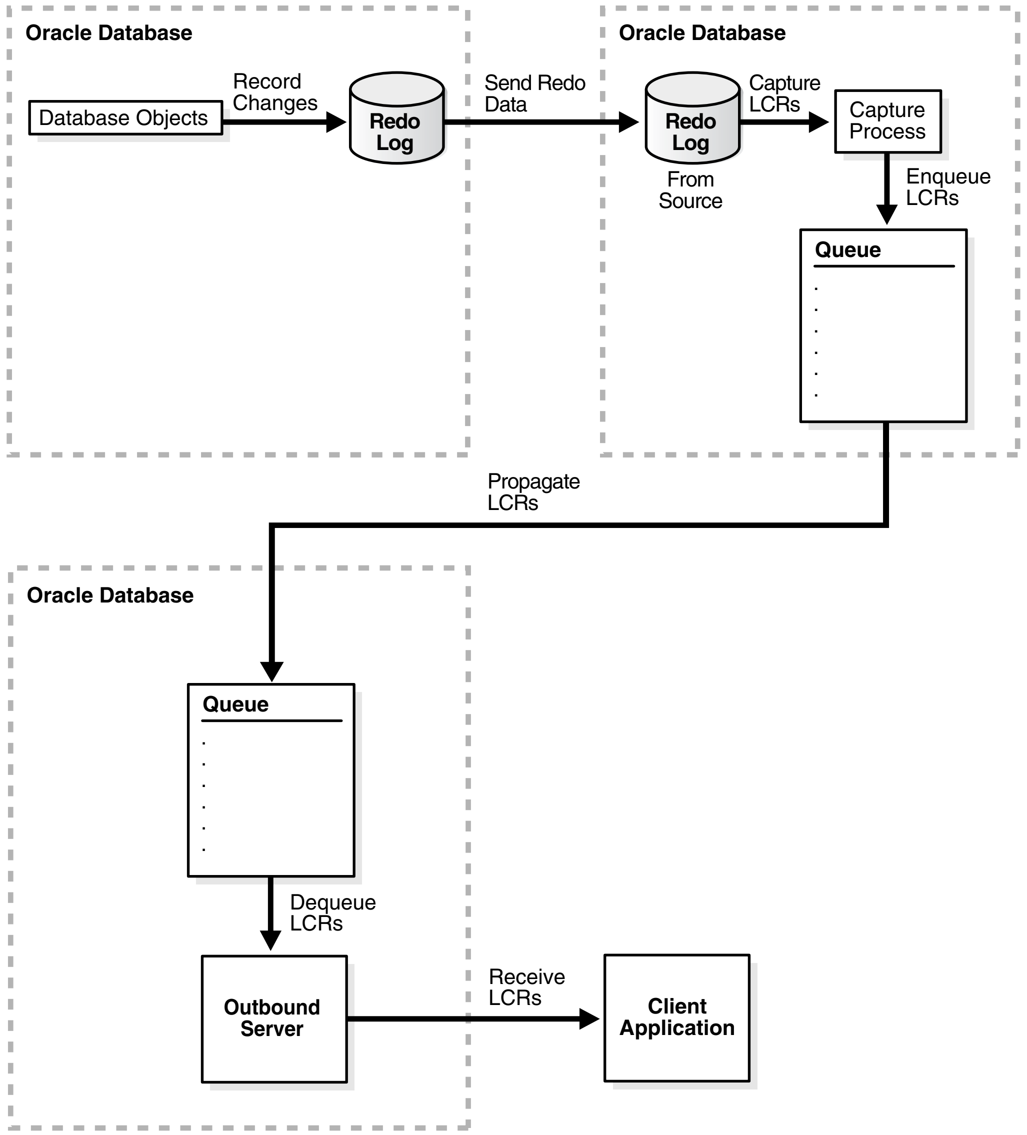


Figure 4–4 Downstream Capture and Outbound Server in Different Databases



If you decide to configure a downstream capture process, then you must decide which type of downstream capture process you want to configure. The following types are available:

- A **real-time downstream capture process** configuration means that redo transport services use the log writer process (LGWR) at the source database to send redo data to the downstream database, and a remote file server process (RFS) at the downstream database receives the redo data over the network and stores the redo data in the standby redo log.
- An **archived-log downstream capture process** configuration means that archived redo log files from the source database are copied to the downstream database, and the capture process captures changes in these archived redo log files. These log files can be transferred automatically using redo transport services, or they can be transferred manually using a method such as FTP.

The advantage of real-time downstream capture over archived-log downstream capture is that real-time downstream capture reduces the amount of time required to capture changes made to the source database. The time is reduced because the real-time downstream capture process does not need to wait for the redo log file to be archived before it can capture changes from it. You can configure multiple real-time downstream capture processes that capture changes from the same source database, but you cannot configure real-time downstream capture for multiple source databases at one downstream database.

The advantage of archived-log downstream capture over real-time downstream capture is that archived-log downstream capture allows downstream capture processes for multiple source databases at a downstream database. You can copy redo log files from multiple source databases to a single downstream database and configure multiple archived-log downstream capture processes to capture changes in these redo log files.

See Also: *Oracle Streams Concepts and Administration*

Prerequisites for Configuring XStream Out

Preparing for an XStream Out outbound server is similar to preparing for an Oracle Streams replication environment. The components used in an Oracle Streams replication environment to capture changes and send them to an apply process are the same components used to capture changes and send them to an outbound server. These components include a capture process and one or more queues. If the capture process runs on a different database than the outbound server, then a propagation is also required.

Several of the tasks described in this section are described in more detail in *Oracle Streams Replication Administrator's Guide*. This section provides an overview of each task and specific information about completing the task for an XStream Out configuration.

Ensure that the following prerequisites are met before configuring XStream Out:

- Configure an XStream Administrator on All Databases
- If Required, Configure Network Connectivity and Database Links
- Ensure That Each Source Database Is in ARCHIVELOG Mode
- Set the Relevant Initialization Parameters
- Configure the Oracle Streams Pool
- If Required, Configure Log File Transfer to a Downstream Database
- If Required, Add Standby Redo Logs for Real-Time Downstream Capture

Configure an XStream Administrator on All Databases To configure and manage an XStream Out configuration, create an XStream administrator on each Oracle database that is involved in the XStream Out configuration.

See Also: "Granting Privileges for the XStream Administrator" on page 4-1

If Required, Configure Network Connectivity and Database Links Network connectivity and database links are not required when all of the components run on the same database. These components include the capture process, queue, and outbound server.

You must configure network connectivity and database links if you decided to configure XStream in either of the following ways:

- The capture process and the outbound server will run on different databases.

- Downstream capture will be used.

See "Decide How to Configure XStream" on page 4-6 for more information about these decisions.

If network connectivity is required, then configure your network and Oracle Net so that the databases can communicate with each other.

The following database links are required:

- When the capture process runs on a different database from the outbound server, create a database link from the capture database to the outbound server database. A propagation uses this database link to send changes from the capture database to the outbound server database.
- When you use downstream capture, create a database link from the capture database to the source database. The source database is the database that generates the redo data that the capture process uses to capture changes. The capture process uses this database link to perform administrative tasks at the source database.

The name of each database link must match the global name of the destination database, and each database link should be created in the XStream administrator's schema.

For example, assume that you want to create a database link in a configuration with the following characteristics:

- The global name of the source database is `dbs1.example.com`.
- The global name of the destination database is `dbs2.example.com`.
- The XStream administrator is `xstrmadmin` at each database.

Given these assumptions, the following statement creates a database link from `dbs1.example.com` to `dbs2.example.com`:

```
CONNECT xstrmadmin@dbs1.example.com
Enter password: password

CREATE DATABASE LINK dbs2.example.com CONNECT TO xstrmadmin
   IDENTIFIED BY password USING 'dbs2.example.com';
```

See Also:

- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about creating database links using Oracle Enterprise Manager
- *Oracle Database Administrator's Guide* for more information about database links

Ensure That Each Source Database Is in ARCHIVELOG Mode Each source database that generates changes that will be captured by a capture process must be in `ARCHIVELOG` mode. For downstream capture processes, the downstream database also must be in `ARCHIVELOG` mode if you plan to configure a real-time downstream capture process. The downstream database does not need to be in `ARCHIVELOG` mode if you plan to run only archived-log downstream capture processes on it.

If you are configuring XStream in an Oracle Real Application Clusters (Oracle RAC) environment, then the archived redo log files of all threads from all instances must be available to any instance running a capture process. This requirement pertains to both local and downstream capture processes.

> **See Also:** *Oracle Database Administrator's Guide* for instructions about running a database in `ARCHIVELOG` mode

Set the Relevant Initialization Parameters Some initialization parameters are important for the configuration, operation, reliability, and performance of the components in an XStream configuration. Set these parameters appropriately.

Oracle Streams Replication Administrator's Guide contains detailed information about all of the initialization parameters that are important for an Oracle Streams environment. The guidelines for setting these parameters also apply to an XStream configuration. In addition to the requirements described in *Oracle Streams Replication Administrator's Guide* for all Oracle Streams components, the following requirements apply to XStream outbound servers:

- Ensure that the `PROCESSES` initialization parameter is set to a value large enough to accommodate the outbound server background processes and all of the other Oracle Database background processes.
- Ensure that the `SESSIONS` initialization parameter is set to a value large enough to accommodate the sessions used by the outbound server background processes and all of the other Oracle Database sessions.

Configure the Oracle Streams Pool The Oracle Streams pool is a portion of memory in the System Global Area (SGA) that is used by Oracle Streams. The Oracle Streams pool stores buffered queue messages in memory, and it provides memory for capture processes and outbound servers. The Oracle Streams pool always stores LCRs captured by a capture process, and it stores LCRs and messages that are enqueued into a buffered queue by applications. Ensure that there is enough space in the Oracle Streams pool at each database to store LCRs and run the components properly.

Each outbound server requires 1 MB of memory. The Oracle Streams pool is initialized the first time an outbound server is started.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for information about Oracle Streams pool requirements

If Required, Configure Log File Transfer to a Downstream Database If you decided to use a local capture process, then log file transfer is not required. However, if you decided to use downstream capture that uses redo transport services to transfer archived redo log files to the downstream database automatically, then configure log file transfer from the source database to the capture database. See "Decide How to Configure XStream" on page 4-6 for information about this decision.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for instructions

If Required, Add Standby Redo Logs for Real-Time Downstream Capture If you decided to configure real-time downstream capture, then add standby redo logs to the capture database. See "Decide How to Configure XStream" on page 4-6 for information about this decision.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for instructions

Preparing for XStream In

Ensure that the following prerequisites are met before configuring XStream In:

- Configure an XStream Administrator
- Set the Relevant Initialization Parameters Relevant
- Configure the Oracle Streams Pool

Configure an XStream Administrator

To configure and manage an XStream In configuration, create an XStream administrator on the Oracle database that will run the XStream inbound server.

> **See Also:** "Granting Privileges for the XStream Administrator" on page 4-1

Set the Relevant Initialization Parameters Relevant

Some initialization parameters are important for the configuration, operation, reliability, and performance of XStream inbound servers. Set these parameters appropriately.

Oracle Streams Replication Administrator's Guide contains detailed information about all of the initialization parameters that are important for an Oracle Streams environment. The guidelines for setting these parameters also apply to an XStream configuration. In addition to the requirements described in *Oracle Streams Replication Administrator's Guide* for all Oracle Streams components, the following requirements apply to XStream inbound servers:

- Ensure that the `PROCESSES` initialization parameter is set to a value large enough to accommodate the inbound server background processes and all of the other Oracle Database background processes.
- Ensure that the `SESSIONS` initialization parameter is set to a value large enough to accommodate the sessions used by the inbound server background processes and all of the other Oracle Database sessions.

Configure the Oracle Streams Pool

The Oracle Streams pool is a portion of memory in the System Global Area (SGA) that provides memory for inbound servers. Ensure that there is enough space in the Oracle Streams pool for the inbound server to run properly. An inbound server requires 1 MB for each inbound server parallelism. For example, if parallelism is set to 10 for an inbound server, then at least 10 MB of memory is required for the inbound server. The Oracle Streams pool also must have enough space to store LCRs before they are applied. The Oracle Streams pool is initialized the first time an inbound server is started.

> **See Also:** *Oracle Streams Replication Administrator's Guide*

Configuring XStream Out

An outbound server in an XStream Out configuration streams Oracle database changes to a client application. The client application attaches to the outbound server using the Oracle Call Interface (OCI) or Java interface to receive these changes.

Configuring an outbound server involves creating the components that send captured database changes to the outbound server. It also involves configuring the outbound server itself, which includes specifying the connect user that the client application will use to attach to the outbound server.

This section contains these topics:

- Configuring an XStream Outbound Server
- Adding an Additional Outbound Server to a Capture Process Stream

Configuring an XStream Outbound Server

You can create an outbound server using the following procedures in the `DBMS_XSTREAM_ADM` package:

- The `CREATE_OUTBOUND` procedure creates an outbound server, a queue, and a capture process in a single database with one procedure call.
- The `ADD_OUTBOUND` procedure only creates an outbound server. You must create the capture process and queue separately, and they must exist before you run the `ADD_OUTBOUND` procedure. You can configure the capture process on the same database as the outbound server or on a different database.

In both cases, you must create the client application that communicates with the outbound server and receives LCRs from the outbound server.

If you require multiple outbound servers, then you can use the `CREATE_OUTBOUND` procedure to create the capture process that captures database changes for the first outbound server. Next, you can run the `ADD_OUTBOUND` procedure to add additional outbound servers that receive the same captured changes. The capture process can reside on the same database as the outbound servers or on a different database.

This section contains these topics:

- Configuring Multiple XStream Out Components Using CREATE_OUTBOUND
- Configuring an Outbound Server Using ADD_OUTBOUND

Configuring Multiple XStream Out Components Using CREATE_OUTBOUND

The `CREATE_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package creates a capture process, queue, and outbound server in a single database. Both the capture process and the outbound server use the queue created by the procedure. When you run the procedure, you provide the name of the new outbound server, while the procedure generates a name for the capture process and queue. If you want all of the components to run on the same database, then the `CREATE_OUTBOUND` procedure is the fastest and easiest way to create an outbound server.

Prerequisites

Before configuring XStream Out, ensure that the following prerequisites are met:

- Complete the tasks described in "Prerequisites for Configuring XStream Out" on page 4-11.

Assumptions

This section makes the following assumptions:

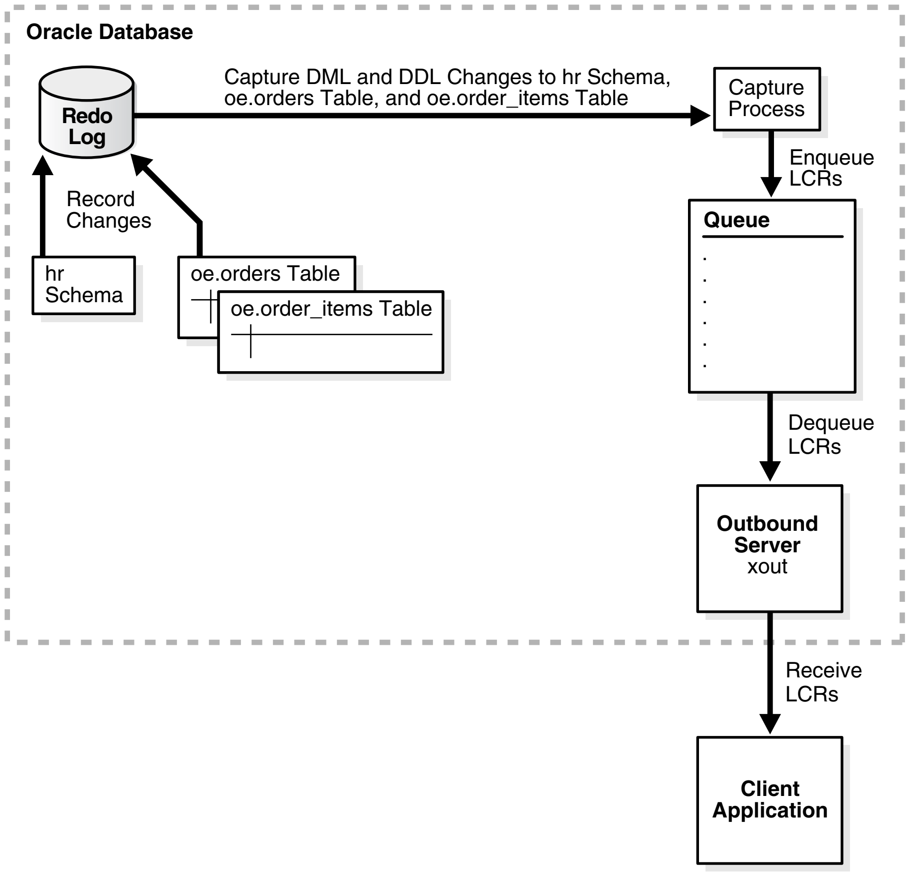
- The capture process will be a local capture process, and it will run on the same database as the outbound server.

  The instructions in this section can only set up the local capture and outbound server on the same database configuration described in "Decide How to Configure XStream" on page 4-6.

- The name of the outbound server is `xout`.
- Data manipulation language (DML) and data definition language (DDL) changes made to the `oe.orders` and `oe.order_items` tables are sent to the outbound server.
- DML and DDL changes made to the `hr` schema are sent to the outbound server.

Figure 4–5 provides an overview of this XStream Out configuration.

Figure 4–5 Sample XStream Out Configuration Created Using CREATE_OUTBOUND



To create an outbound server using the `CREATE_OUTBOUND` procedure:

1. In SQL*Plus, connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `CREATE_OUTBOUND` procedure.

   Given the assumptions for this section, run the following `CREATE_OUTBOUND` procedure:

```
DECLARE
  tables  DBMS_UTILITY.UNCL_ARRAY;
  schemas DBMS_UTILITY.UNCL_ARRAY;
  BEGIN
    tables(1)  := 'oe.orders';
    tables(2)  := 'oe.order_items';
    schemas(1) := 'hr';
  DBMS_XSTREAM_ADM.CREATE_OUTBOUND(
    server_name     =>  'xout',
    table_names     =>  tables,
    schema_names    =>  schemas);
END;
/
```

Running this procedure performs the following actions:

- Configures supplemental logging for the `oe.orders` and `oe.order_items` tables and for all of the tables in the `hr` schema.
- Creates a queue with a system-generated name that is used by the capture process and the outbound server.
- Creates and starts a capture process with a system-generated name with rule sets that instruct it to capture DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema.
- Creates and starts an outbound server named `xout` with rule sets that instruct it to send DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema to the client application.
- Sets the current user as the connect user for the outbound server. In this example, the current user is the XStream administrator. The client application must connect to the database as the connect user to interact with the outbound server.

Tip: To capture and send all database changes to the outbound server, specify `NULL` (the default) for the `table_names` and `schema_names` parameters.

3. Create and run the client application that will connect to the outbound server and receive the LCRs. See "Sample XStream Client Application" on page 4-26 for a sample application.
4. To add one or more additional outbound servers that receive LCRs from the capture process created in Step 2, follow the instructions in "Adding an Additional Outbound Server to a Capture Process Stream" on page 4-22.

See Also: "CREATE_OUTBOUND Procedure" on page 8-21

Configuring an Outbound Server Using ADD_OUTBOUND

The `ADD_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package creates an outbound server. This procedure does not create the capture process or the queue. You must configure these components manually.

The instructions in this section can set up any of the configurations described in "Decide How to Configure XStream" on page 4-6. However, if you chose the local capture and outbound server on the same database configuration, then it is usually easier to use the `CREATE_OUTBOUND` procedure to configure all of the components simultaneously. See "Configuring Multiple XStream Out Components Using CREATE_OUTBOUND" on page 4-15.

Prerequisites

Before configuring XStream Out, ensure that the following prerequisites are met:

- Complete the tasks described in "Prerequisites for Configuring XStream Out" on page 4-11.

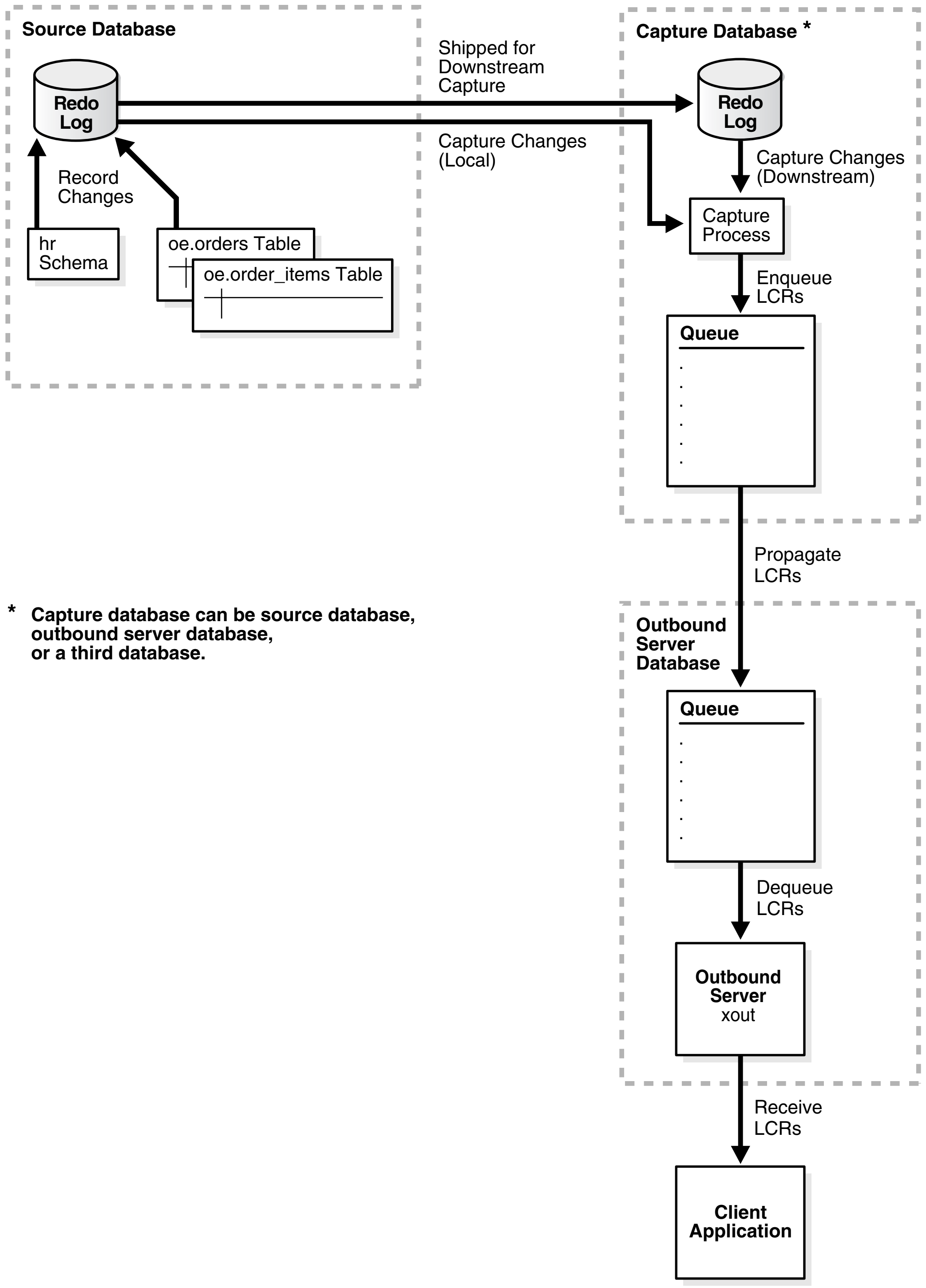
Assumptions

This section makes the following assumptions:

- The name of the outbound server is `xout`.
- The queue used by the outbound server is `xstrmadmin.xstream_queue`.
- The source database is `db1.example.com`.
- DML and DDL changes made to the `oe.orders` and `oe.order_items` tables are sent to the outbound server.
- DML and DDL changes made to the `hr` schema are sent to the outbound server.
- The capture process for the outbound server does not exist. (If the capture process exists, then skip Steps 1 to 3, and go to Step 4.)

Figure 4–6 provides an overview of this XStream Out configuration.

Figure 4–6 Sample XStream Out Configuration Created Using ADD_OUTBOUND



Note: If the capture database and the outbound server database are different databases, then a propagation propagates the LCRs to a queue at the outbound server database. If the capture database and outbound server database are the same, then the propagation is not needed.

To create an outbound server using the `ADD_OUTBOUND` procedure:

1. In SQL*Plus, connect to the database that will run the capture process (the capture database) as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Create the queue that will be used by the capture process.

   See *Oracle Streams Replication Administrator's Guide* for instructions.

3. Create the capture process.

   Add rules to the capture process's rule sets to capture changes to the `hr` schema, the `oe.orders` table, and the `oe.order_items` table. Do not start the capture process.

   See *Oracle Streams Replication Administrator's Guide* for instructions.

4. If the capture process will run on a different database than the outbound server, then set the `xout_client_exists` capture process parameter to `Y`.

   Setting this parameter to `Y` enables the capture process to send LCRs to an outbound server.

   Skip this step if the capture process will run on the same database as the outbound server. In this case, the `xout_client_exists` capture process parameter will be set to `Y` automatically.

   See *Oracle Streams Concepts and Administration* for information about setting a capture process parameter. See *Oracle Database PL/SQL Packages and Types Reference* for information about the `xout_client_exists` capture process parameter.

5. Connect to the source database.

   The source database is the database that contains the database objects for which the capture process will capture changes. The source database and the capture database might be the same database.

6. Ensure that required supplemental logging is specified for the database objects at the source database.

   Supplemental logging is required for the database objects for which the capture process will capture changes. If the capture database and the source database are the same database, then supplemental logging might have been specified during capture process creation.

   Ensure that the following supplemental logging is specified at the source database:

   - Any columns at the source database that are used in a primary key in tables for which changes are processed by the outbound server must be unconditionally logged in a log group or by database supplemental logging of primary key columns.
   - Any columns at the source database that are used by a rule or a rule-based transformation must be unconditionally logged.

   For the example in this section, ensure that supplemental logging is configured for the `hr` schema, the `oe.orders` table, and the `oe.order_items` table.

   See *Oracle Streams Replication Administrator's Guide* for instructions about specifying supplemental logging.

7. Connect to the database that will run the outbound server as the XStream administrator.

8. Create the queue that will be used by the outbound server.

   This step is not required if the capture process and the outbound server run on the same database and use the same queue.

   See *Oracle Streams Replication Administrator's Guide* for instructions.

9. Run the `ADD_OUTBOUND` procedure.

   Given the assumption for this section, run the following `ADD_OUTBOUND` procedure:

```
DECLARE
  tables  DBMS_UTILITY.UNCL_ARRAY;
  schemas DBMS_UTILITY.UNCL_ARRAY;
  BEGIN
    tables(1)  := 'oe.orders';
    tables(2)  := 'oe.order_items';
    schemas(1) := 'hr';
  DBMS_XSTREAM_ADM.ADD_OUTBOUND(
    server_name     =>  'xout',
    queue_name      =>  'xstrmadmin.xstream_queue',
    source_database =>  'db1.example.com',
    table_names     =>  tables,
    schema_names    =>  schemas);
END;
/
```

   If the capture process runs on the same database as the outbound server, then specify the capture process's queue for the `queue_name` parameter.

   Running this procedure performs the following actions:

   - Creates an outbound server named `xout`. The outbound server has rule sets that instruct it to send DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema to the client application. The rules specify that these changes must have originated at the `db1.example.com` database. The outbound server dequeues LCRs from the queue `xstrmadmin.xstream_queue`.
   - Sets the current user as the connect user for the outbound server. In this example, the current user is the XStream administrator. The client application must connect to the database as the connect user to interact with the outbound server.

   Tip: For the outbound server to receive all of the LCRs sent by the capture process, specify `NULL` (the default) for the `table_names` and `schema_names` parameters.

10. Connect to the capture database as the XStream administrator.

11. Create the propagation that sends LCRs from the capture process's queue on the local database to the queue used by the outbound server on the outbound server database.

    Add rules to the propagation's rule sets to send changes to the `hr` schema, the `oe.orders` table, and the `oe.order_items` table from the source queue to the destination queue.

    This step is not required if the capture process and the outbound server run on the same database and use the same queue.

    See *Oracle Streams Replication Administrator's Guide* for instructions.

12. Create and run the client application that will connect to the outbound server and receive the LCRs. See "Sample XStream Client Application" on page 4-26 for a sample application.
13. Connect to the database that is running outbound server as the XStream administrator.
14. Start the outbound server if it is disabled. For example:

    ```
    exec DBMS_APPLY_ADM.START_APPLY('xout');
    ```

15. Connect to the capture database as the XStream administrator.
16. Start the capture process created in Step 3.

    See *Oracle Streams Concepts and Administration* for instructions.
17. To add one or more additional outbound servers that receive LCRs from the capture process created in Step 3, follow the instructions in "Adding an Additional Outbound Server to a Capture Process Stream" on page 4-22.

See Also: "ADD_OUTBOUND Procedure" on page 8-7

Adding an Additional Outbound Server to a Capture Process Stream

XStream Out configurations often require multiple outbound servers that process a stream of LCRs from a single capture process. This section describes adding an additional outbound server to a database that already includes at least one outbound server. The additional outbound server uses the same queue as another outbound server to receive the LCRs from the capture process. When an XStream Out environment exists, use the `ADD_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package to add another outbound server to a capture process stream.

Prerequisites

Before completing the steps in this section, configure an XStream Out environment that includes at least one outbound server. The following sections describe configuring and XStream Out environment:

- "Configuring Multiple XStream Out Components Using CREATE_OUTBOUND" on page 4-15
- "Configuring an Outbound Server Using ADD_OUTBOUND" on page 4-17

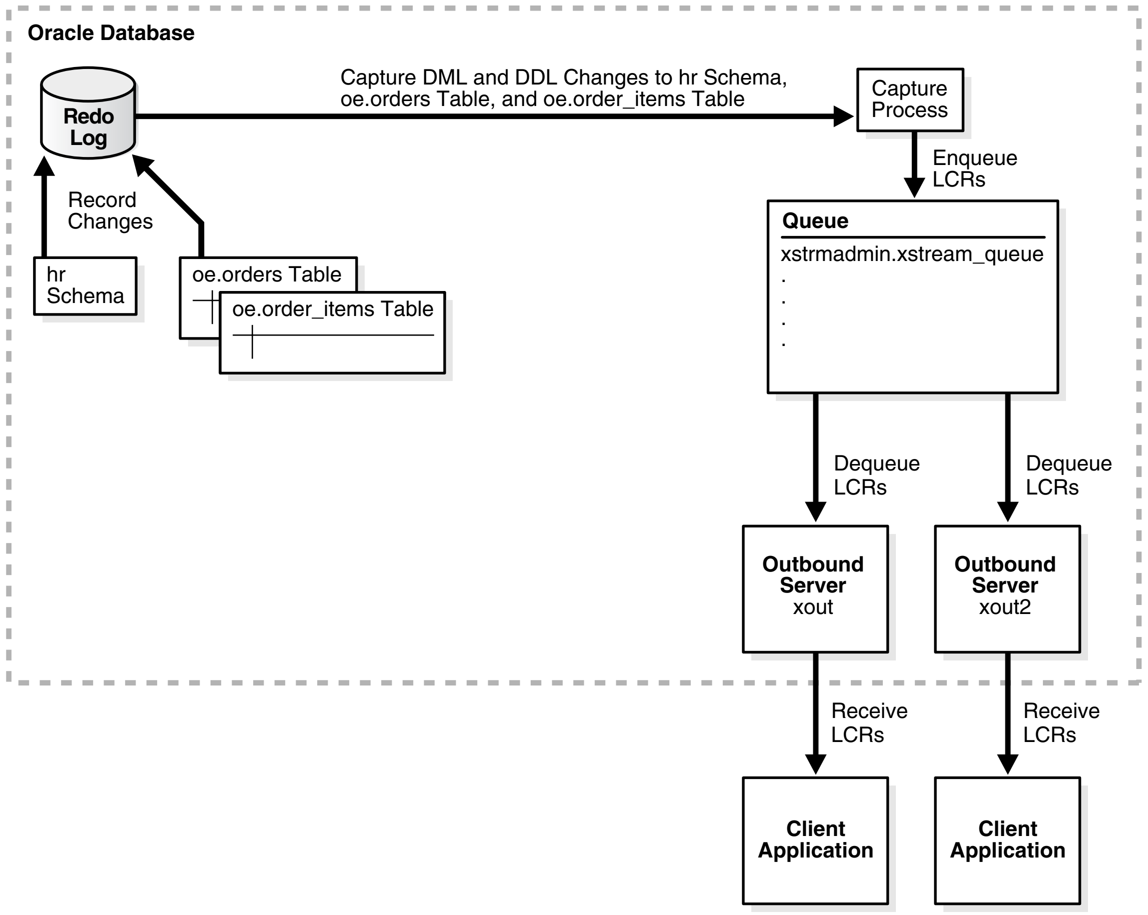
Assumptions

This section makes the following assumptions:

- The name of the outbound server is `xout2`.
- The queue used by the outbound server is `xstrmadmin.xstream_queue`.
- DML and DDL changes made to the `oe.orders` and `oe.order_items` tables are sent to the outbound server.
- DML and DDL changes made to the `hr` schema are sent to the outbound server.
- The source database for the database changes is `db1.example.com`.

Figure 4–7 provides an overview of this XStream Out configuration.

Figure 4–7 Sample XStream Out Configuration With an Additional Outbound Server



To add another outbound server to a capture process stream using the `ADD_OUTBOUND` procedure:

1. In SQL*Plus, connect to the database that will run the additional outbound server as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Determine the name of the queue used by an existing outbound server that receives LCRs from the capture process.

   Run the query in "Displaying General Information About an Outbound Server" on page 6-3 to determine the owner and name of the queue. This query also shows the name of the capture process and the source database name.

3. Run the `ADD_OUTBOUND` procedure.

   Given the assumptions for this section, run the following `ADD_OUTBOUND` procedure:

```
DECLARE
  tables  DBMS_UTILITY.UNCL_ARRAY;
  schemas DBMS_UTILITY.UNCL_ARRAY;
  BEGIN
    tables(1)  := 'oe.orders';
    tables(2)  := 'oe.order_items';
    schemas(1) := 'hr';
  DBMS_XSTREAM_ADM.ADD_OUTBOUND(
    server_name     =>  'xout2',
    queue_name      =>  'xstrmadmin.xstream_queue',
    source_database =>  'db1.example.com',
    table_names     =>  tables,
    schema_names    =>  schemas);
END;
/
```

Running this procedure performs the following actions:

- Creates an outbound server named `xout2`. The outbound server has rule sets that instruct it to send DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema to the client application. The rules specify that these changes must have originated at the `db1.example.com` database. The outbound server dequeues LCRs from the queue `xstrmadmin.xstream_queue`.
- Sets the current user as the connect user for the outbound server. In this example, the current user is the XStream administrator. The client application must connect to the database as the connect user to interact with the outbound server.

Tip: For the outbound server to receive all of the LCRs sent by the capture process, specify `NULL` (the default) for the `table_names` and `schema_names` parameters.

4. If a client application does not exist, then create and run the client application that will connect to the outbound server and receive the LCRs. See "Sample XStream Client Application" on page 4-26 for a sample application.

See Also: "ADD_OUTBOUND Procedure" on page 8-7

Configuring XStream In

An inbound server in an XStream In configuration receives a stream of changes from a client application. The inbound server can apply these changes to database objects in an Oracle database, or it can process the changes in a customized way. A client application can attach to an inbound server and send row changes and DDL changes encapsulated in LCRs using the OCI or Java interface.

The `CREATE_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package creates an inbound server. You must create the client application that communicates with the inbound server and sends LCRs to the inbound server.

Prerequisites

Before configuring XStream In, ensure that the following prerequisites are met:

- Complete the tasks described in "Preparing for XStream In" on page 4-14.

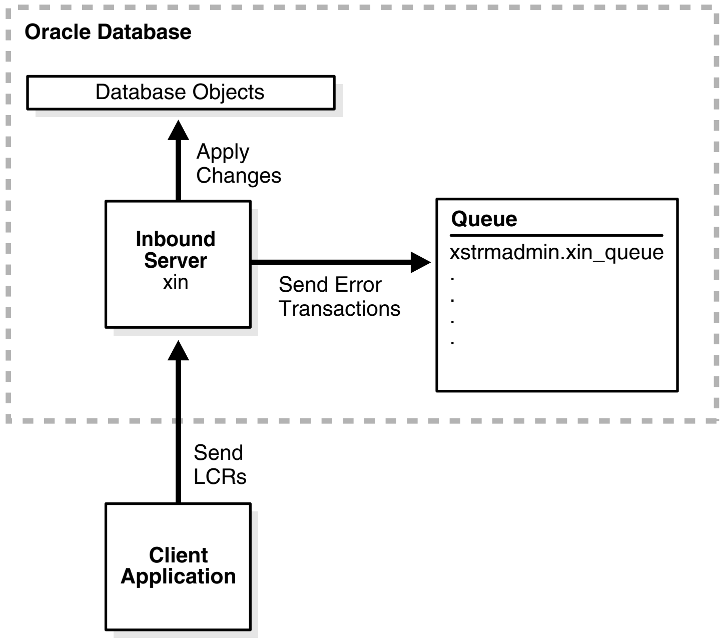
Assumptions

This section makes the following assumptions:

- The name of the inbound server is `xin`.
- The inbound server applies all of the changes it receives from the XStream client application.
- The queue used by the inbound server is `xstrmadmin.xin_queue`.

Figure 4–8 provides an overview of this XStream In configuration.

Figure 4–8 Sample XStream In Configuration



To create an inbound server:

1. In SQL*Plus, connect to the database that will run the inbound server as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `CREATE_INBOUND` procedure.

   For example, the following `CREATE_INBOUND` procedure configures an inbound server named `xin`:

```
BEGIN
  DBMS_XSTREAM_ADM.CREATE_INBOUND(
    server_name => 'xin',
    queue_name  => 'xin_queue');
END;
/
```

   Running this procedure performs the following actions:

   - Creates an inbound server named `xin`.
   - Sets the queue with the name `xin_queue` as the inbound server's queue, and creates this queue if it does not exist. This queue does not store LCRs sent by the client application. Instead, it stores error transactions if an LCR raises an error. The current user is the queue owner. In this example, the current user is the XStream administrator.
   - Sets the current user as the apply user for the inbound server. In this example, the current user is the XStream administrator. The client application must connect to the database as the apply user to interact with the inbound server.

Tip: By default, an inbound server does not use rules or rule sets. Therefore, it processes all LCRs sent to it by the client application. To add rules and rule sets, use the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. See *Oracle Streams Concepts and Administration*.

3. If necessary, create apply handlers for the inbound server.

   Apply handlers are optional. Apply handlers process LCRs sent to an inbound server in a customized way.

   See Also: *Oracle Streams Concepts and Administration*

4. Create and run the client application that will connect to the inbound server and send LCRs to it.

   See "Sample XStream Client Application" on page 4-26 for a sample application.

5. If the inbound server is disabled, then start the inbound server.

   For example, enter the following:

   ```
   exec DBMS_APPLY_ADM.START_APPLY('xin');
   ```

See Also: "CREATE_INBOUND Procedure" on page 8-19

Sample XStream Client Application

This section contains a sample XStream client application. This application illustrates the basic tasks that are required of an XStream Out and XStream In application.

The application performs the following tasks:

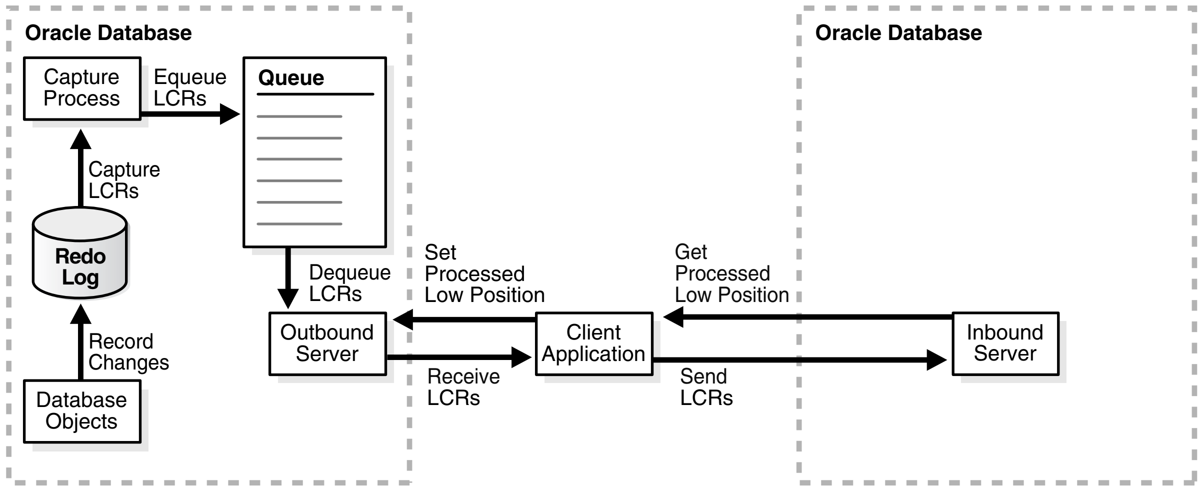
- It attaches to an XStream outbound server and inbound server and waits for LCRs from the outbound server. The outbound server and inbound server are in two different databases.
- When it receives an LCR from the outbound server, it immediately sends the LCR to the inbound server.
- It periodically gets the processed low position from the inbound server and sends this value to the outbound server.
- It periodically sends a "ping" LCR from the outbound server to the inbound server to move the inbound server's processed low position forward in times of low activity.

In an XStream Out configuration that does not send LCRs to an inbound server, the client application must obtain the processed low position in another way.

This application waits indefinitely for transactions from the outbound server. To interrupt the application, enter the interrupt command for your operating system. For example, the interrupt command on some operating systems is `control-C`. If the program is restarted, then the outbound server starts sending LCRs from the processed low position that was set during the previous run.

Figure 4–9 provides an overview of the XStream environment configured in this section.

Figure 4–9 Sample XStream Configuration



Before running the sample application, ensure that the following components exist:

- Two Oracle databases with network connectivity between them
- An XStream administrator on both databases
- An outbound server configuration on one database, including a capture process, queue, and outbound server
- An inbound server configuration on another database

The sample applications in the following sections perform the same tasks. One sample application uses the OCI API, and the other uses the Java API.

- Sample XStream Client Application for the Oracle Call Interface API
- Sample XStream Client Application for the Java API

Note: An Oracle Database installation includes several XStream demos. These demos are in the following location:

```
$ORACLE_HOME/rdbms/demo/xstream
```

See Also:

- "Position of LCRs and XStream In" on page 2-13
- "Configuring XStream Out" on page 4-14
- "Configuring XStream In" on page 4-24
- Part IV, "XStream OCI API Reference"
- *Oracle Database XStream Java API Reference*
- "Granting Privileges for the XStream Administrator" on page 4-1

Sample XStream Client Application for the Oracle Call Interface API

To run the sample XStream client application for the OCI API, compile and link the application file, and enter the following on a command line:

```
xio -ob_svr xout_name -ob_db sn_xout_db -ob_usr xout_cu -ob_pwd xout_cu_pass
-ib_svr xin_name -ib_db sn_xin_db -ib_usr xin_au -ib_pwd xin_au_pass
```

Substitute the appropriate values for the following placeholders:

- *xout_name* is the name of the outbound server.
- *sn_xout_db* is the service name for the outbound server's database.
- *xout_cu* is the outbound server's connect user.
- *xout_cu_pass* is the password for the outbound server's connect user.
- *xin_name* is the name of the inbound server.
- *sn_xin_db* is the service name for the inbound server's database.
- *xin_au* is the inbound server's apply user.
- *xin_au_pass* is the password for the inbound server's apply user.

When the sample client application is running, it prints information about the row LCRs it is processing. The output looks similar to the following:

```
 ----------- ROW LCR Header  -----------------
  src_db_name=DB.EXAMPLE.COM
  cmd_type=UPDATE txid=17.0.74
  owner=HR oname=COUNTRIES

 ----------- ROW LCR Header  -----------------
  src_db_name=DB.EXAMPLE.COM
  cmd_type=COMMIT txid=17.0.74

 ----------- ROW LCR Header  -----------------
  src_db_name=DB.EXAMPLE.COM
  cmd_type=UPDATE txid=12.25.77
  owner=OE oname=ORDERS

 ----------- ROW LCR Header  -----------------
  src_db_name=DB.EXAMPLE.COM
  cmd_type=UPDATE txid=12.25.77
  owner=OE oname=ORDERS
```

This output contains the following information for each row LCR:

- `src_db_name` shows the source database for the change encapsulated in the row LCR.
- `cmd_type` shows the type of SQL statement that made the change.
- `txid` shows the transaction ID of the transaction that includes the row LCR.
- `owner` shows the owner of the database object that was changed.
- `oname` shows the name of the database object that was changed.

This demo is available in the following location:

```
$ORACLE_HOME/rdbms/demo/xstream/oci
```

The file name for the demo is `xio.c`. See the `README.txt` file in the demo directory for more information about compiling and running the application.

The code for the sample application that uses the OCI API follows:

```
#ifndef OCI_ORACLE
#include <oci.h>
```

```
#endif

#ifndef _STDIO_H
#include <stdio.h>
#endif

#ifndef _STDLIB_H
#include <stdlib.h>
#endif

#ifndef _STRING_H
#include <string.h>
#endif

#ifndef _MALLOC_H
#include <malloc.h>
#endif

/*----------------------------------------------------------------------
 *           Internal structures
 *----------------------------------------------------------------------*/

#define M_DBNAME_LEN    (128)

typedef struct conn_info                                   /* connect info */
{
  oratext * user;
  ub4       userlen;
  oratext * passw;
  ub4       passwlen;
  oratext * dbname;
  ub4       dbnamelen;
  oratext * svrnm;
  ub4       svrnmlen;
} conn_info_t;

typedef struct params
{
  conn_info_t  xout;                                      /* outbound info */
  conn_info_t  xin;                                        /* inbound info */
} params_t;

typedef struct oci                                          /* OCI handles */
{
  OCIEnv      *envp;                               /* Environment handle */
  OCIError    *errp;                                     /* Error handle */
  OCIServer   *srvp;                                    /* Server handle */
  OCISvcCtx   *svcp;                                   /* Service handle */
  OCISession  *authp;
  OCIStmt    *stmtp;
  boolean     attached;
  boolean     outbound;
} oci_t;

static void connect_db(conn_info_t *opt_params_p, oci_t ** ocip, ub2 char_csid,
                       ub2 nchar_csid);
static void disconnect_db(oci_t * ocip);
static void ocierror(oci_t * ocip, char * msg);
static void attach(oci_t * ocip, conn_info_t *conn, boolean outbound);
static void detach(oci_t *ocip);
static void get_lcrs(oci_t *xin_ocip, oci_t *xout_ocip);
static void get_chunks(oci_t *xin_ocip, oci_t *xout_ocip);
static void print_lcr(oci_t *ocip, void *lcrp, ub1 lcrtype,
                      oratext **src_db_name, ub2  *src_db_namel);
static void print_chunk (ub1 *chunk_ptr, ub4 chunk_len, ub2 dty);
static void get_inputs(conn_info_t *xout_params, conn_info_t *xin_params,
                       int argc, char ** argv);
static void get_db_charsets(conn_info_t *params_p, ub2 *char_csid,
                            ub2 *nchar_csid);
static void set_client_charset(oci_t *outbound_ocip);

#define OCICALL(ocip, function) do {\
sword status=function;\
if (OCI_SUCCESS==status) break;\
else if (OCI_ERROR==status) \
{ocierror(ocip, (char *)"OCI_ERROR");\
exit(1);}\
else {printf("Error encountered %d\n", status);\
exit(1);}\
} while(0)

/*----------------------------------------------------------------------
 *                M A I N   P R O G R A M
 *---------------------------------------------------------------------*/
main(int argc, char **argv)
{
  /* Outbound and inbound connection info */
  conn_info_t   xout_params;
  conn_info_t   xin_params;
  oci_t        *xout_ocip = (oci_t *)NULL;
  oci_t        *xin_ocip = (oci_t *)NULL;
  ub2           obdb_char_csid = 0;                /* outbound db char csid */
  ub2           obdb_nchar_csid = 0;              /* outbound db nchar csid */

  /* parse command line arguments */
  get_inputs(&xout_params, &xin_params, argc, argv);

  /* Get the outbound database CHAR and NCHAR character set info */
  get_db_charsets(&xout_params, &obdb_char_csid, &obdb_nchar_csid);

  /* Connect to the outbound db and set the client env to the outbound charsets
   * to minimize character conversion when transferring LCRs from outbound
   * directly to inbound server.
   */
  connect_db(&xout_params, &xout_ocip, obdb_char_csid, obdb_nchar_csid);

  /* Attach to outbound server */
  attach(xout_ocip, &xout_params, TRUE);

  /* connect to inbound db and set the client charsets the same as the
   * outbound db charsets.
   */
  connect_db(&xin_params, &xin_ocip, obdb_char_csid, obdb_nchar_csid);

  /* Attach to inbound server */
  attach(xin_ocip, &xin_params, FALSE);

  /* Get lcrs from outbound server and send to inbound server */
  get_lcrs(xin_ocip, xout_ocip);
  /* Detach from XStream servers */
  detach(xout_ocip);
  detach(xin_ocip);

  /* Disconnect from both databases */
  disconnect_db(xout_ocip);
  disconnect_db(xin_ocip);

  free(xout_ocip);
  free(xin_ocip);
  exit (0);
}

/*---------------------------------------------------------------------
 * connect_db - Connect to the database and set the env to the given
 * char and nchar character set ids.
 *---------------------------------------------------------------------*/
static void connect_db(conn_info_t *params_p, oci_t **ociptr, ub2 char_csid,
                ub2 nchar_csid)
{
  oci_t        *ocip;

  printf ("Connect to Oracle as %.*s@%.*s ",
          params_p->userlen, params_p->user,
          params_p->dbnamelen, params_p->dbname);

  if (char_csid && nchar_csid)
    printf ("using char csid=%d and nchar csid=%d", char_csid, nchar_csid);

  printf("\n");

  ocip = (oci_t *)malloc(sizeof(oci_t));

  if (OCIEnvNlsCreate(&ocip->envp, OCI_OBJECT, (dvoid *)0,
                     (dvoid * (*)(dvoid *, size_t)) 0,
                     (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                     (void (*)(dvoid *, dvoid *)) 0,
                     (size_t) 0, (dvoid **) 0, char_csid, nchar_csid))
  {
    ocierror(ocip, (char *)"OCIEnvCreate() failed");
  }

  if (OCIHandleAlloc((dvoid *) ocip->envp, (dvoid **) &ocip->errp,
                     (ub4) OCI_HTYPE_ERROR, (size_t) 0, (dvoid **) 0))
  {
    ocierror(ocip, (char *)"OCIHandleAlloc(OCI_HTYPE_ERROR) failed");
  }

  /* Logon to database */
  OCICALL(ocip,
          OCILogon(ocip->envp, ocip->errp, &ocip->svcp,
                   params_p->user, params_p->userlen,
                   params_p->passw, params_p->passwlen,
                   params_p->dbname, params_p->dbnamelen));

  /* allocate the server handle */
  OCICALL(ocip,
          OCIHandleAlloc((dvoid *) ocip->envp, (dvoid **) &ocip->srvp,
                         OCI_HTYPE_SERVER, (size_t) 0, (dvoid **) 0));
  OCICALL(ocip,
          OCIHandleAlloc((dvoid *) ocip->envp, (dvoid **) &ocip->stmtp,
                     (ub4) OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));

  if (*ociptr == (oci_t *)NULL)
  {
    *ociptr = ocip;
  }
}

/*---------------------------------------------------------------------
 * get_db_charsets - Get the database CHAR and NCHAR character set ids.
 *---------------------------------------------------------------------*/
static const oratext GET_DB_CHARSETS[] =  \
 "select parameter, value from nls_database_parameters where parameter = \
 'NLS_CHARACTERSET' or parameter = 'NLS_NCHAR_CHARACTERSET'";

#define PARM_BUFLEN      (30)

static void get_db_charsets(conn_info_t *params_p, ub2 *char_csid,
                            ub2 *nchar_csid)
{
  OCIDefine  *defnp1 = (OCIDefine *) NULL;
  OCIDefine  *defnp2 = (OCIDefine *) NULL;
  oratext     parm[PARM_BUFLEN];
  oratext     value[OCI_NLS_MAXBUFSZ];
  ub2         parm_len = 0;
  ub2         value_len = 0;
  oci_t       ocistruct;
  oci_t      *ocip = &ocistruct;

  *char_csid = 0;
  *nchar_csid = 0;
  memset (ocip, 0, sizeof(ocistruct));

  if (OCIEnvCreate(&ocip->envp, OCI_OBJECT, (dvoid *)0,
                   (dvoid * (*)(dvoid *, size_t)) 0,
                   (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                   (void (*)(dvoid *, dvoid *)) 0,
                   (size_t) 0, (dvoid **) 0))
  {
    ocierror(ocip, (char *)"OCIEnvCreate() failed");
  }

  if (OCIHandleAlloc((dvoid *) ocip->envp, (dvoid **) &ocip->errp,
                     (ub4) OCI_HTYPE_ERROR, (size_t) 0, (dvoid **) 0))
  {
    ocierror(ocip, (char *)"OCIHandleAlloc(OCI_HTYPE_ERROR) failed");
  }

  OCICALL(ocip,
          OCILogon(ocip->envp, ocip->errp, &ocip->svcp,
                   params_p->user, params_p->userlen,
                   params_p->passw, params_p->passwlen,
                   params_p->dbname, params_p->dbnamelen));

  OCICALL(ocip,
          OCIHandleAlloc((dvoid *) ocip->envp, (dvoid **) &ocip->stmtp,
                     (ub4) OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));
  /* Execute stmt to select the db nls char and nchar character set */
  OCICALL(ocip,
          OCIStmtPrepare(ocip->stmtp, ocip->errp,
                         (CONST text *)GET_DB_CHARSETS,
                         (ub4)strlen((char *)GET_DB_CHARSETS),
                         (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  OCICALL(ocip,
          OCIDefineByPos(ocip->stmtp, &defnp1,
                         ocip->errp, (ub4) 1, parm,
                         PARM_BUFLEN, SQLT_CHR, (void*) 0,
                         &parm_len, (ub2 *)0, OCI_DEFAULT));

  OCICALL(ocip,
          OCIDefineByPos(ocip->stmtp, &defnp2,
                         ocip->errp, (ub4) 2, value,
                         OCI_NLS_MAXBUFSZ, SQLT_CHR, (void*) 0,
                         &value_len, (ub2 *)0, OCI_DEFAULT));

  OCICALL(ocip,
          OCIStmtExecute(ocip->svcp, ocip->stmtp,
                         ocip->errp, (ub4)0, (ub4)0,
                         (const OCISnapshot *)0,
                         (OCISnapshot *)0, (ub4)OCI_DEFAULT));

  while (OCIStmtFetch(ocip->stmtp, ocip->errp, 1,
                      OCI_FETCH_NEXT, OCI_DEFAULT) == OCI_SUCCESS)
  {
    value[value_len] = '\0';
    if (parm_len == strlen("NLS_CHARACTERSET") &&
        !memcmp(parm, "NLS_CHARACTERSET", parm_len))
    {
      *char_csid = OCINlsCharSetNameToId(ocip->envp, value);
      printf("Outbound database NLS_CHARACTERSET = %.*s (csid = %d) \n",
             value_len, value, *char_csid);
    }
    else if (parm_len == strlen("NLS_NCHAR_CHARACTERSET") &&
             !memcmp(parm, "NLS_NCHAR_CHARACTERSET", parm_len))
    {
      *nchar_csid = OCINlsCharSetNameToId(ocip->envp, value);
      printf("Outbound database NLS_NCHAR_CHARACTERSET = %.*s (csid = %d) \n",
             value_len, value, *nchar_csid);
    }
  }

  disconnect_db(ocip);
}

/*----------------------------------------------------------------------
 * attach - Attach to XStream server specified in connection info
 *----------------------------------------------------------------------*/
static void attach(oci_t * ocip, conn_info_t *conn, boolean outbound)
{
  sword       err;

  printf ("Attach to XStream %s server '%.*s'\n",
          outbound ? "outbound" : "inbound",
          conn->svrnmlen, conn->svrnm);

  if (outbound)
```

```
    {
      OCICALL(ocip,
              OCIXStreamOutAttach(ocip->svcp, ocip->errp, conn->svrnm,
                                  (ub2)conn->svrnmlen, (ub1 *)0, 0, OCI_DEFAULT));
    }
    else
    {
      OCICALL(ocip,
              OCIXStreamInAttach(ocip->svcp, ocip->errp, conn->svrnm,
                                 (ub2)conn->svrnmlen,
                                 (oratext *)"From_XOUT", 9,
                                 (ub1 *)0, 0, OCI_DEFAULT));
    }

    ocip->attached = TRUE;
    ocip->outbound = outbound;
  }

  /*---------------------------------------------------------------------
   * ping_svr - Ping inbound server by sending a commit LCR.
   *---------------------------------------------------------------------*/
  static void ping_svr(oci_t *xin_ocip, void *commit_lcr,
                       ub1 *cmtpos, ub2 cmtpos_len,
                       oratext *source_db, ub2 source_db_len)
  {
    OCIDate     src_time;
    oratext     txid[128];

    OCICALL(xin_ocip, OCIDateSysDate(xin_ocip->errp, &src_time));
    sprintf(txid, "Ping %2d:%2d:%2d",
            src_time.OCIDateTime.OCITimeHH,
            src_time.OCIDateTime.OCITimeMI,
            src_time.OCIDateTime.OCITimeSS);

    /* Initialize LCR with new txid and commit position */
    OCICALL(xin_ocip,
            OCILCRHeaderSet(xin_ocip->svcp, xin_ocip->errp,
                            source_db, source_db_len,
                            (oratext *)OCI_LCR_ROW_CMD_COMMIT,
                            (ub2)strlen(OCI_LCR_ROW_CMD_COMMIT),
                            (oratext *)0, 0,                     /* null owner */
                            (oratext *)0, 0,                    /* null object */
                            (ub1 *)0, 0,                           /* null tag */
                            txid, (ub2)strlen((char *)txid),
                            &src_time, cmtpos, cmtpos_len,
                            0, commit_lcr, OCI_DEFAULT));

    /* Send commit lcr to inbound server. */
    if (OCIXStreamInLCRSend(xin_ocip->svcp, xin_ocip->errp, commit_lcr,
                            OCI_LCR_XROW, 0, OCI_DEFAULT) == OCI_ERROR)
    {
      ocierror(xin_ocip, (char *)"OCIXStreamInLCRSend failed in ping_svr()");
    }
  }

  /*---------------------------------------------------------------------
   * get_lcrs - Get LCRs from outbound server and send to inbound server.
   *---------------------------------------------------------------------*/
  static void get_lcrs(oci_t *xin_ocip, oci_t *xout_ocip)
  {
```

```
sword       status = OCI_SUCCESS;
void       *lcr;
ub1         lcrtype;
oraub8      flag;
ub1         proclwm[OCI_LCR_MAX_POSITION_LEN];
ub2         proclwm_len = 0;
ub1         sv_pingpos[OCI_LCR_MAX_POSITION_LEN];
ub2         sv_pingpos_len = 0;
ub1         fetchlwm[OCI_LCR_MAX_POSITION_LEN];
ub2         fetchlwm_len = 0;
void       *commit_lcr = (void *)0;
oratext    *lcr_srcdb = (oratext *)0;
ub2         lcr_srcdb_len = 0;
oratext     source_db[M_DBNAME_LEN];
ub2         source_db_len = 0;
ub4         lcrcnt = 0;

/* create an lcr to ping the inbound server periodically by sending a
 * commit lcr.
 */
commit_lcr = (void*)0;
OCICALL(xin_ocip,
        OCILCRNew(xin_ocip->svcp, xin_ocip->errp, OCI_DURATION_SESSION,
                  OCI_LCR_XROW, &commit_lcr, OCI_DEFAULT));

while (status == OCI_SUCCESS)
{
  lcrcnt = 0;                         /* reset lcr count before each batch */

  while ((status =
            OCIXStreamOutLCRReceive(xout_ocip->svcp, xout_ocip->errp,
                                    &lcr, &lcrtype, &flag,
                                    fetchlwm, &fetchlwm_len, OCI_DEFAULT))
                                           == OCI_STILL_EXECUTING)
  {
    lcrcnt++;

    /* print header of LCR just received */
    print_lcr(xout_ocip, lcr, lcrtype, &lcr_srcdb, &lcr_srcdb_len);

    /* save the source db to construct ping lcr later */
    if (!source_db_len && lcr_srcdb_len)
    {
      memcpy(source_db, lcr_srcdb, lcr_srcdb_len);
      source_db_len = lcr_srcdb_len;
    }

    /* send the LCR just received */
    if (OCIXStreamInLCRSend(xin_ocip->svcp, xin_ocip->errp,
                            lcr, lcrtype, flag, OCI_DEFAULT) == OCI_ERROR)
    {
      ocierror(xin_ocip, (char *)"OCIXStreamInLCRSend failed");
    }

    /* If LCR has chunked columns (i.e, has LOB/Long/XMLType columns) */
    if (flag & OCI_XSTREAM_MORE_ROW_DATA)
    {
      /* receive and send chunked columns */
      get_chunks(xin_ocip, xout_ocip);
    }
    }

    if (status == OCI_ERROR)
      ocierror(xout_ocip, (char *)"OCIXStreamOutLCRReceive failed");

    /* clear the saved ping position if we just received some new lcrs */
    if (lcrcnt)
    {
      sv_pingpos_len = 0;
    }

    /* If no lcrs received during previous WHILE loop and got a new fetch
     * LWM then send a commit lcr to ping the inbound server with the new
     * fetch LWM position.
     */
    else if (fetchlwm_len > 0 && source_db_len > 0 &&
        (fetchlwm_len != sv_pingpos_len ||
         memcmp(sv_pingpos, fetchlwm, fetchlwm_len)))
    {
      /* To ensure we don't send multiple lcrs with duplicate position, send
       * a new ping only if we have saved the last ping position.
       */
      if (sv_pingpos_len > 0)
      {
        ping_svr(xin_ocip, commit_lcr, fetchlwm, fetchlwm_len,
                 source_db, source_db_len);
      }

      /* save the position just sent to inbound server */
      memcpy(sv_pingpos, fetchlwm, fetchlwm_len);
      sv_pingpos_len = fetchlwm_len;
    }

    /* flush inbound network to flush all lcrs to inbound server */
    OCICALL(xin_ocip,
            OCIXStreamInFlush(xin_ocip->svcp, xin_ocip->errp, OCI_DEFAULT));


    /* get processed LWM of inbound server */
    OCICALL(xin_ocip,
            OCIXStreamInProcessedLWMGet(xin_ocip->svcp, xin_ocip->errp,
                                        proclwm, &proclwm_len, OCI_DEFAULT));

    if (proclwm_len > 0)
    {
      /* Set processed LWM for outbound server */
      OCICALL(xout_ocip,
              OCIXStreamOutProcessedLWMSet(xout_ocip->svcp, xout_ocip->errp,
                                           proclwm, proclwm_len, OCI_DEFAULT));
    }
  }

  if (status != OCI_SUCCESS)
    ocierror(xout_ocip, (char *)"get_lcrs() encounters error");
}

/*----------------------------------------------------------------------
 * get_chunks - Get each chunk for the current LCR and send it to
 *              the inbound server.
 *---------------------------------------------------------------------*/
```

```
static void get_chunks(oci_t *xin_ocip, oci_t *xout_ocip)
{
  oratext *colname;
  ub2      colname_len;
  ub2      coldty;
  oraub8   col_flags;
  ub2      col_csid;
  ub4      chunk_len;
  ub1     *chunk_ptr;
  oraub8   row_flag;
  sword    err;
  sb4      rtncode;

  do
  {
    /* Get a chunk from outbound server */
    OCICALL(xout_ocip,
            OCIXStreamOutChunkReceive(xout_ocip->svcp, xout_ocip->errp,
                                      &colname, &colname_len, &coldty,
                                      &col_flags, &col_csid, &chunk_len,
                                      &chunk_ptr, &row_flag, OCI_DEFAULT));

    /* print chunked column info */
    printf(
      "  Chunked column name=%.*s DTY=%d  chunk len=%d csid=%d col_flag=0x%x\n",
      colname_len, colname, coldty, chunk_len, col_csid, col_flags);

    /* print chunk data */
    print_chunk(chunk_ptr, chunk_len, coldty);

    /* Send the chunk just received to inbound server */
    OCICALL(xin_ocip,
            OCIXStreamInChunkSend(xin_ocip->svcp, xin_ocip->errp, colname,
                                  colname_len, coldty, col_flags,
                                  col_csid, chunk_len, chunk_ptr,
                                  row_flag, OCI_DEFAULT));

  } while (row_flag & OCI_XSTREAM_MORE_ROW_DATA);
}

/*---------------------------------------------------------------------
 * print_chunk - Print chunked column information. Only print the first
 *               50 bytes for each chunk.
 *---------------------------------------------------------------------*/
static void print_chunk (ub1 *chunk_ptr, ub4 chunk_len, ub2 dty)
{
#define MAX_PRINT_BYTES     (50)          /* print max of 50 bytes per chunk */

  ub4  print_bytes;

  if (chunk_len == 0)
    return;

  print_bytes = chunk_len > MAX_PRINT_BYTES ? MAX_PRINT_BYTES : chunk_len;

  printf("  Data = ", chunk_len);
  if (dty == SQLT_CHR)
    printf("%.*s", print_bytes, chunk_ptr);
  else
  {
    ub2  idx;

    for (idx = 0; idx < print_bytes; idx++)
      printf("%02x", chunk_ptr[idx]);
  }
  printf("\n");
}

/*----------------------------------------------------------------------
 * print_lcr - Print header information of given lcr.
 *----------------------------------------------------------------------*/
static void print_lcr(oci_t *ocip, void *lcrp, ub1 lcrtype,
                      oratext **src_db_name, ub2  *src_db_namel)
{
  oratext     *cmd_type;
  ub2          cmd_type_len;
  oratext     *owner;
  ub2          ownerl;
  oratext     *oname;
  ub2          onamel;
  oratext     *txid;
  ub2          txidl;
  sword        ret;

  printf("\n ----------- %s LCR Header  -----------------\n",
         lcrtype == OCI_LCR_XDDL ? "DDL" : "ROW");

  /* Get LCR Header information */
  ret = OCILCRHeaderGet(ocip->svcp, ocip->errp,
                        src_db_name, src_db_namel,              /* source db */
                        &cmd_type, &cmd_type_len,            /* command type */
                        &owner, &ownerl,                       /* owner name */
                        &oname, &onamel,                      /* object name */
                        (ub1 **)0, (ub2 *)0,                      /* lcr tag */
                        &txid, &txidl, (OCIDate *)0,   /* txn id  & src time */
                        (ub2 *)0, (ub2 *)0,              /* OLD/NEW col cnts */
                        (ub1 **)0, (ub2 *)0,                 /* LCR position */
                        (oraub8*)0, lcrp, OCI_DEFAULT);

  if (ret != OCI_SUCCESS)
    ocierror(ocip, (char *)"OCILCRHeaderGet failed");
  else
  {
    printf("  src_db_name=%.*s\n  cmd_type=%.*s txid=%.*s\n",
           *src_db_namel, *src_db_name, cmd_type_len, cmd_type, txidl, txid );

    if (ownerl > 0)
      printf("  owner=%.*s oname=%.*s \n", ownerl, owner, onamel, oname);
  }
}

/*----------------------------------------------------------------------
 * detach - Detach from XStream server
 *----------------------------------------------------------------------*/
static void detach(oci_t * ocip)
{
  sword  err = OCI_SUCCESS;

  printf ("Detach from XStream %s server\n",
          ocip->outbound ? "outbound" : "inbound" );
  if (ocip->outbound)
  {
    OCICALL(ocip, OCIXStreamOutDetach(ocip->svcp, ocip->errp, OCI_DEFAULT));
  }
  else
  {
    OCICALL(ocip, OCIXStreamInDetach(ocip->svcp, ocip->errp,
                                    (ub1 *)0, (ub2 *)0,    /* processed LWM */
                                    OCI_DEFAULT));
  }
}

/*---------------------------------------------------------------------
 * disconnect_db  - Logoff from the database
 *---------------------------------------------------------------------*/
static void disconnect_db(oci_t * ocip)
{
  if (OCILogoff(ocip->svcp, ocip->errp))
  {
    ocierror(ocip, (char *)"OCILogoff() failed");
  }

  if (ocip->errp)
    OCIHandleFree((dvoid *) ocip->errp, (ub4) OCI_HTYPE_ERROR);

  if (ocip->envp)
    OCIHandleFree((dvoid *) ocip->envp, (ub4) OCI_HTYPE_ENV);
}

/*---------------------------------------------------------------------
 * ocierror - Print error status and exit program
 *---------------------------------------------------------------------*/
static void ocierror(oci_t * ocip, char * msg)
{
  sb4 errcode=0;
  text bufp[4096];

  if (ocip->errp)
  {
    OCIErrorGet((dvoid *) ocip->errp, (ub4) 1, (text *) NULL, &errcode,
                bufp, (ub4) 4096, (ub4) OCI_HTYPE_ERROR);
    printf("%s\n%s", msg, bufp);
  }
  else
    puts(msg);

  printf ("\n");
  exit(1);
}

/*--------------------------------------------------------------------
 * print_usage - Print command usage
 *---------------------------------------------------------------------*/
static void print_usage(int exitcode)
{
  puts("\nUsage: xio -ob_svr <outbound_svr> -ob_db <outbound_db>\n"
       "           -ob_usr <conn_user> -ob_pwd <conn_user_pwd>\n"
       "           -ib_svr <inbound_svr> -ib_db <inbound_db>\n"
       "           -ib_usr <apply_user> -ib_pwd <apply_user_pwd>\n");
  puts("  ob_svr  : outbound server name\n"
       "  ob_db   : database name of outbound server\n"
       "  ob_usr  : connect user to outbound server\n"
       "  ob_pwd  : password of outbound's connect user\n"
       "  ib_svr  : inbound server name\n"
       "  ib_db   : database name of inbound server\n"
       "  ib_usr  : apply user for inbound server\n"
       "  ib_pwd  : password of inbound's apply user\n");

  exit(exitcode);
}

/*----------------------------------------------------------------------
 * get_inputs - Get user inputs from command line
 *---------------------------------------------------------------------*/
static void get_inputs(conn_info_t *xout_params, conn_info_t *xin_params,
                       int argc, char ** argv)
{
  char * option;
  char * value;

  memset (xout_params, 0, sizeof(*xout_params));
  memset (xin_params, 0, sizeof(*xin_params));
  while(--argc)
  {
    /* get the option name */
    argv++;
    option = *argv;

    /* check that the option begins with a "-" */
    if (!strncmp(option, (char *)"-", 1))
    {
      option ++;
    }
    else
    {
      printf("Error: bad argument '%s'\n", option);
      print_usage(1);
    }

    /* get the value of the option */
    --argc;
    argv++;

    value = *argv;

    if (!strncmp(option, (char *)"ob_db", 5))
    {
      xout_params->dbname = (oratext *)value;
      xout_params->dbnamelen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ob_usr", 6))
    {
      xout_params->user = (oratext *)value;
      xout_params->userlen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ob_pwd", 6))
    {
      xout_params->passw = (oratext *)value;
      xout_params->passwlen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ob_svr", 6))
    {
      xout_params->svrnm = (oratext *)value;
      xout_params->svrnmlen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_db", 5))
    {
      xin_params->dbname = (oratext *)value;
      xin_params->dbnamelen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_usr", 6))
    {
      xin_params->user = (oratext *)value;
      xin_params->userlen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_pwd", 6))
    {
      xin_params->passw = (oratext *)value;
      xin_params->passwlen = strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_svr", 6))
    {
      xin_params->svrnm = (oratext *)value;
      xin_params->svrnmlen = strlen(value);
    }
    else
    {
      printf("Error: unknown option '%s'.\n", option);
      print_usage(1);
    }
  }

  /* print usage and exit if any argument is not specified */
  if (!xout_params->svrnmlen || !xout_params->passwlen ||
      !xout_params->userlen || !xout_params->dbnamelen ||
      !xin_params->svrnmlen || !xin_params->passwlen ||
      !xin_params->userlen || !xin_params->dbnamelen)
  {
    printf("Error: missing command arguments. \n");
    print_usage(1);
  }
}
```

Sample XStream Client Application for the Java API

To run the sample XStream client application for the Java API, compile and link the application file, and enter the following on a command line:

```
java xio xsin_oraclesid xsin_host xsin_port xsin_username
xsin_passwd xin_servername xsout_oraclesid xsout_host xsout_port
xsout_username xsout_passwd xsout_servername
```

Substitute the appropriate values for the following placeholders:

- *xsin_oraclesid* is the Oracle SID of the inbound server's database.
- *xsin_host* is the host name of the computer system running the inbound server.
- *xsin_port* is the port number of the listener for the inbound server's database.

- *xsin_username* is the inbound server's apply user.
- *xsin_passwd* is the password for the inbound server's apply user.
- *xin_servername* is the name of the inbound server.
- *xsout_oraclesid* is the Oracle SID of the outbound server's database.
- *xsout_host* is the host name of the computer system running the outbound server.
- *xsout_port* is the port number of the listener for the outbound server's database.
- *xsout_username* is the outbound server's connect user.
- *xsout_passwd* is the password for the outbound server's connect user.
- *xsout_servername* is the name of the outbound server.

When the sample client application is running, it prints information about attaching to the inbound server and outbound server, along with the last position for each server. The output looks similar to the following:

```
xsin_host = server2.example.com
xsin_port = 1482
xsin_ora_sid = db2
xsin connection url: jdbc:oracle:oci:@server2.example.com:1482:db2
xsout_host = server1.example.com
xsout_port = 1481
xsout_ora_sid = db1
xsout connection url: jdbc:oracle:oci:@server1.example.com:1481:db1
Attached to inbound server:xin
Inbound Server Last Position is:
0000000920250000000010000000100000009202500000000100000000101
Attached to outbound server:xout
Last Position is: 0000000920250000000010000000100000009202500000000100000000101
```

This demo is available in the following location:

```
$ORACLE_HOME/rdbms/demo/xstream/java
```

The file name for the demo is `xio.java`. See the `README.txt` file in the demo directory for more information about compiling and running the application.

The code for the sample application that uses the Java API follows:

```
import oracle.streams.*;
import oracle.jdbc.internal.OracleConnection;
import oracle.jdbc.*;
import oracle.sql.*;
import java.sql.*;
import java.util.*;

public class xio
{
  public static String xsinusername = null;
  public static String xsinpasswd = null;
  public static String xsinName = null;
  public static String xsoutusername = null;
  public static String xsoutpasswd = null;
  public static String xsoutName = null;
  public static String in_url = null;
  public static String out_url = null;
  public static Connection in_conn = null;
  public static Connection out_conn = null;
  public static XStreamIn xsIn = null;
  public static XStreamOut xsOut = null;
  public static byte[] lastPosition = null;
  public static byte[] processedLowPosition = null;

  public static void main(String args[])
  {
    // get connection url to inbound and outbound server
    in_url = parseXSInArguments(args);
    out_url = parseXSOutArguments(args);

    // create connection to inbound and outbound server
    in_conn = createConnection(in_url, xsinusername, xsinpasswd);
    out_conn = createConnection(out_url, xsoutusername, xsoutpasswd);

    // attach to inbound and outbound server
    xsIn = attachInbound(in_conn);
    xsOut = attachOutbound(out_conn);

    // main loop to get lcrs
    get_lcrs(xsIn, xsOut);

    // detach from inbound and outbound server
    detachInbound(xsIn);
    detachOutbound(xsOut);
  }

  // parse the arguments to get the conncetion url to inbound db
  public static String parseXSInArguments(String args[])
  {
    String trace, pref;
    String orasid, host, port;

    if (args.length != 12)
    {
      printUsage();
      System.exit(0);
    }

    orasid = args[0];
    host = args[1];
    port = args[2];
    xsinusername = args[3];
    xsinpasswd = args[4];
    xsinName = args[5];

    System.out.println("xsin_host = "+host);
    System.out.println("xsin_port = "+port);
    System.out.println("xsin_ora_sid = "+orasid);

    String in_url = "jdbc:oracle:oci:@"+host+":"+port+":"+orasid;
    System.out.println("xsin connection url: "+ in_url);

    return in_url;
  }

  // parse the arguments to get the conncetion url to outbound db
  public static String parseXSOutArguments(String args[])
  {
    String trace, pref;
    String orasid, host, port;
    if (args.length != 12)
    {
      printUsage();
      System.exit(0);
    }

    orasid = args[6];
    host = args[7];
    port = args[8];
    xsoutusername = args[9];
    xsoutpasswd = args[10];
    xsoutName = args[11];


    System.out.println("xsout_host = "+host);
    System.out.println("xsout_port = "+port);
    System.out.println("xsout_ora_sid = "+orasid);

    String out_url = "jdbc:oracle:oci:@"+host+":"+port+":"+orasid;
    System.out.println("xsout connection url: "+ out_url);

    return out_url;
  }

  // print out sample program usage message
  public static void printUsage()
  {
    System.out.println("");
    System.out.println("Usage: java xio "+"<xsin_oraclesid> " + "<xsin_host> "
                                         + "<xsin_port> ");
    System.out.println("                "+"<xsin_username> " + "<xsin_passwd> "
                                         + "<xsin_servername> ");
    System.out.println("                "+"<xsout_oraclesid> " + "<xsout_host> "
                                         + "<xsout_port> ");
    System.out.println("                "+"<xsout_username> " + "<xsout_passwd> "
                                         + "<xsout_servername> ");
  }

  // create a connection to an Oracle Database
  public static Connection createConnection(String url,
                                            String username,
                                            String passwd)
  {
    try
    {
      DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
      return DriverManager.getConnection(url, username, passwd);
    }
    catch(Exception e)
    {
      System.out.println("fail to establish DB connection to: " +url);
      e.printStackTrace();
      return null;
    }
  }

  // attach to the XStream Inbound Server
  public static XStreamIn attachInbound(Connection in_conn)
  {
    XStreamIn xsIn = null;
    try
    {
      xsIn = XStreamIn.attach((OracleConnection)in_conn, xsinName,
                              "XSDEMOINCLIENT" , XStreamIn.DEFAULT_MODE);

      // use last position to decide where should we start sending LCRs
      lastPosition = xsIn.getLastPosition();
      System.out.println("Attached to inbound server:"+xsinName);
      System.out.print("Inbound Server Last Position is: ");
      if (null == lastPosition)
      {
        System.out.println("null");
      }
      else
      {
        printHex(lastPosition);
      }
      return xsIn;
    }
    catch(Exception e)
    {
      System.out.println("cannot attach to inbound server: "+xsinName);
      System.out.println(e.getMessage());
      e.printStackTrace();
      return null;
    }
  }

  // attach to the XStream Outbound Server
  public static XStreamOut attachOutbound(Connection out_conn)
  {
    XStreamOut xsOut = null;

    try
    {
      // when attach to an outbound server, client needs to tell outbound
      // server the last position.
      xsOut = XStreamOut.attach((OracleConnection)out_conn, xsoutName,
                                lastPosition, XStreamOut.DEFAULT_MODE);
      System.out.println("Attached to outbound server:"+xsoutName);
      System.out.print("Last Position is: ");
      if (lastPosition != null)
      {
        printHex(lastPosition);
      }
      else
      {
        System.out.println("NULL");
      }
      return xsOut;
    }
    catch(Exception e)
    {
      System.out.println("cannot attach to outbound server: "+xsoutName);
      System.out.println(e.getMessage());
      e.printStackTrace();
      return null;
    }
  }
```

```
// detach from the XStream Inbound Server
public static void detachInbound(XStreamIn xsIn)
{
  byte[] processedLowPosition = null;
  try
  {
    processedLowPosition = xsIn.detach(XStreamIn.DEFAULT_MODE);
    System.out.print("Inbound server processed low Position is: ");
    if (processedLowPosition != null)
    {
      printHex(processedLowPosition);
    }
    else
    {
      System.out.println("NULL");
    }
  }
  catch(Exception e)
  {
    System.out.println("cannot detach from the inbound server: "+xsinName);
    System.out.println(e.getMessage());
    e.printStackTrace();
  }
}

// detach from the XStream Outbound Server
public static void detachOutbound(XStreamOut xsOut)
{
  try
  {
    xsOut.detach(XStreamOut.DEFAULT_MODE);
  }
  catch(Exception e)
  {
    System.out.println("cannot detach from the outbound server: "+xsoutName);
    System.out.println(e.getMessage());
    e.printStackTrace();
  }
}

public static void get_lcrs(XStreamIn xsIn, XStreamOut xsOut)
{
  byte[] ping_pos = null;
  byte[] fetchlwm = null;
  String src_db = null;

  if (null == xsIn)
  {
    System.out.println("xstreamIn is null");
    System.exit(0);
  }

  if (null == xsOut)
  {
    System.out.println("xstreamOut is null");
    System.exit(0);
  }

  try
```

```
{
  while(true)
  {
    // receive an LCR from outbound server
    LCR alcr = xsOut.receiveLCR(XStreamOut.DEFAULT_MODE);
    fetchlwm = xsOut.getFetchLowWatermark();

    // save source db for ping lcr
    if (null != alcr)
      src_db = alcr.getSourceDatabaseName();

    if (xsOut.getBatchStatus() == XStreamOut.EXECUTING) // batch is active
    {
      assert alcr != null;
      // send the LCR to the inbound server
      xsIn.sendLCR(alcr, XStreamIn.DEFAULT_MODE);

      // also get chunk data for this LCR if any
      if (alcr instanceof RowLCR)
      {
        // receive chunk from outbound then send to inbound
        if (((RowLCR)alcr).hasChunkData())
        {
          ChunkColumnValue chunk = null;
          do
          {
            chunk = xsOut.receiveChunk(XStreamOut.DEFAULT_MODE);
            xsIn.sendChunk(chunk, XStreamIn.DEFAULT_MODE);
          } while (!chunk.isEndOfRow());
        }
      }
      processedLowPosition = alcr.getPosition();
      ping_pos = processedLowPosition;
    }
    else  // batch is end
    {
      assert alcr == null;

      // send ping lcr if we haven't received any lcr in the batch
      // but we got a new fetch lwm, then send a commit lcr to
      // ping the inbound server with the new fetch LWM position
      if (null != src_db && null != fetchlwm &&
          !samePos(fetchlwm,ping_pos))
      {
        xsIn.sendLCR(createPing(src_db, fetchlwm),
                     XStreamIn.DEFAULT_MODE);
        ping_pos = fetchlwm;
      }

      // flush the network
      xsIn.flush(XStreamIn.DEFAULT_MODE);
      // get the processed_low_position from inbound server
      processedLowPosition =
          xsIn.getProcessedLowWatermark();
      // update the processed_low_position at oubound server
      if (null != processedLowPosition)
        xsOut.setProcessedLowWatermark(processedLowPosition,
                                       XStreamOut.DEFAULT_MODE);
    }
  }
```

```
    }
    catch(Exception e)
    {
      System.out.println("exception when processing LCRs");
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
  }

  public static void printHex(byte[] b)
  {
    for (int i = 0; i < b.length; ++i)
    {
      System.out.print(
        Integer.toHexString((b[i]&0xFF) | 0x100).substring(1,3));
    }
    System.out.println("");
  }

  // ping lcr is used to bump up the inbound server watermark
  private static RowLCR createPing(String src_db, byte[] pos)
  {
    java.util.Date today = new java.util.Date();
    java.sql.Timestamp now = new java.sql.Timestamp(today.getTime());
    oracle.sql.DATE src_time = new oracle.sql.DATE(now);


    RowLCR alcr = new DefaultRowLCR();
    ((RowLCR)alcr).setSourceDatabaseName(src_db);
    ((RowLCR)alcr).setSourceTime(src_time);
    ((RowLCR)alcr).setPosition(pos);
    ((RowLCR)alcr).setCommandType(RowLCR.COMMIT);
    ((RowLCR)alcr).setTransactionId("Ping: " + src_time.toString());

    return alcr;
  }

  private static boolean samePos(byte[] pos1, byte[] pos2)
  {
    int   cmp_len;
    boolean    result;

    if (pos1.length != pos2.length)
      return false;

    for (int i = 0; i<pos1.length; i++)
    {
      if (pos1[i] != pos2[i])
        return false;
    }

    return true;
  }
}
```

5
Managing XStream

This chapter provides instructions for managing XStream.

This chapter contains these topics:

See Also:

- Chapter 2, "XStream Concepts"
- Chapter 3, "XStream Use Cases"
- Chapter 4, "Configuring XStream"
- Chapter 6, "Monitoring XStream"
- Chapter 7, "Troubleshooting XStream"

About Managing XStream

This chapter describes managing an XStream Out configuration and an XStream In configuration. This chapter provides instructions for modifying the database components that are part of an XStream configuration, such as outbound severs, inbound servers, capture processes, and rules.

The main interface for managing XStream database components is PL/SQL. Specifically, use the following Oracle supplied PL/SQL packages to manage XStream:

- `DBMS_XSTREAM_ADM`

  The `DBMS_XSTREAM_ADM` package is the main package for managing XStream. This package includes subprograms that enable you to configure, modify, or drop outbound servers and inbound servers.

  See Chapter 8, "DBMS_XSTREAM_ADM" for detailed information about this package.

- `DBMS_XSTREAM_AUTH`

  The `DBMS_XSTREAM_AUTH` package enables you to configure and modify XStream administrators.

  See Chapter 9, "DBMS_XSTREAM_AUTH" for detailed information about this package.

- `DBMS_APPLY_ADM`

The `DBMS_APPLY_ADM` package enables you modify outbound servers and inbound servers.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about this package.

- `DBMS_CAPTURE_ADM`

  The `DBMS_CAPTURE_ADM` package enables you configure and modify capture processes.

  See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about this package.

- `DBMS_STREAMS_ADM`

  The `DBMS_STREAMS_ADM` package enables you modify the rules used by capture processes, outbound servers, and inbound servers.

  See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about this package.

Managing XStream Out

This section describes managing an XStream Out configuration.

This section contains these topics:

- Checking Whether the DBMS_XSTREAM_ADM Package Can Manage a Capture Process
- Managing Rules for an XStream Out Configuration
- Changing the Connect User for an Outbound Server
- Changing the Capture User of the Capture Process for an Outbound Server
- Changing the Start SCN or Start Time of the Capture Process for an Outbound Server
- Dropping Components in an XStream Out Configuration

Note: With XStream Out, an Oracle Streams apply process functions as an outbound server. Therefore, you can use the instructions for managing an apply process to manage an outbound server. See *Oracle Database 2 Day + Data Replication and Integration Guide* and *Oracle Streams Concepts and Administration*.

See Also:

- "XStream Out" on page 2-1
- "Configuring XStream Out" on page 4-14
- "Monitoring XStream Out" on page 6-3

Checking Whether the DBMS_XSTREAM_ADM Package Can Manage a Capture Process

In some XStream Out configurations, you can use the `DBMS_XSTREAM_ADM` package to manage the capture process that captures changes for an outbound server. However, other configurations require that you use the `DBMS_CAPTURE_ADM` package or the `DBMS_STREAMS_ADM` package to manage the capture process.

Specifically, the DBMS_XSTREAM_ADM package can manage an outbound server's capture process in the following ways:

- Add rules to and remove rules from the capture process's rule sets
- Change the capture user for the capture process
- Drop the capture process

The DBMS_XSTREAM_ADM package can manage an outbound server's capture process in either of the following cases:

- The capture process was created by the CREATE_OUTBOUND procedure in the DBMS_XSTREAM_ADM package.
- The queue used by the capture process was created by the CREATE_OUTBOUND procedure.

To check whether an outbound server's capture process can be managed by the DBMS_XSTREAM_ADM package:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Outbound Server Name' FORMAT A30
COLUMN CAPTURE_NAME HEADING 'Capture Process Name' FORMAT A30

SELECT SERVER_NAME,
       CAPTURE_NAME
  FROM DBA_XSTREAM_OUTBOUND;
```

   Your output looks similar to the following:

```
Outbound Server Name           Capture Process Name
------------------------------ ------------------------------
XOUT                           CAP$_XOUT_4
```

   If the Capture Process Name for an outbound server is non-NULL, then the DBMS_XSTREAM_ADM package can manage the capture process. In this case, you can also manage the capture process using the DBMS_CAPTURE_ADM package or the DBMS_STREAMS_ADM package. However, it is usually better to manage the capture process for an outbound server using the DBMS_XSTREAM_ADM package when it is possible.

   If the Capture Process Name for an outbound server is NULL, then the DBMS_XSTREAM_ADM package cannot manage the capture process. In this case, you must manage the capture process using the DBMS_CAPTURE_ADM package or the DBMS_STREAMS_ADM package.

See Also:

- "ALL_XSTREAM_OUTBOUND" on page 12-8
- *Oracle Streams Concepts and Administration* for information about managing a capture process using the DBMS_CAPTURE_ADM package or the DBMS_STREAMS_ADM package

Managing Rules for an XStream Out Configuration

This section describes managing rules for an XStream Out configuration. Rules control which database changes are streamed to the outbound server and which database changes the outbound server streams to the client application.

This section contains these topics:

- Adding Rules to an XStream Out Configuration
- Removing Rules from an XStream Out Configuration

See Also: *Oracle Streams Concepts and Administration*

Adding Rules to an XStream Out Configuration

This section describes adding schema rules, table rules, and subset rules to an XStream Out configuration.

This section contains these topics:

- Adding Schema Rules and Table Rules to an XStream Out Configuration
- Adding Subset Rules to an Outbound Server's Positive Rule Set

Adding Schema Rules and Table Rules to an XStream Out Configuration This section describes adding schema rules and table rules to an XStream Out configuration using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package. The `ALTER_OUTBOUND` procedure adds rules for both data manipulation language (DML) and data definition language (DDL) changes.

When you follow the instructions in this section, the `ALTER_OUTBOUND` procedure always adds rules for the specified schemas and tables to one of the outbound server's rule sets. If the `DBMS_XSTREAM_ADM` package can manage the outbound server's capture process, then the `ALTER_OUTBOUND` procedure also adds rules for the specified schemas and tables to one of the rule sets used by this capture process.

To determine whether the `DBMS_XSTREAM_ADM` package can manage the outbound server's capture process, see "Checking Whether the DBMS_XSTREAM_ADM Package Can Manage a Capture Process" on page 5-2. If the `DBMS_XSTREAM_ADM` package cannot manage the outbound server's capture process, then the `ALTER_OUTBOUND` procedure adds rules to the outbound server's rule set only. In this case, if rules for same schemas and tables should be added to the capture process's rule set as well, then see *Oracle Streams Concepts and Administration* for instructions about adding them.

In addition, if the capture process is running on a different database than the outbound server, then add schema and table rules to the propagation that sends logical change records (LCRs) to the outbound server's database. See *Oracle Streams Concepts and Administration* for instructions.

To add schema rules and table rules to an XStream Out configuration:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `ALTER_OUTBOUND` procedure, and specify the following parameters:

   - `server_name` - Specify the name of the outbound server.
   - `table_names` - Specify the tables for which to add rules, or specify `NULL` to add no table rules.

- schema_name - Specify the schemas for which to add rules, or specify NULL to add no schema rules.
- add - Specify TRUE so that the rules are added. (Rules are removed if you specify FALSE.)
- inclusion_rule - Specify TRUE to add rules to the positive rule set of the outbound server, or specify FALSE to add rules to the negative rule set of the outbound server. If the DBMS_XSTREAM_ADM package can manage the outbound server's capture process, then rules are also added to this capture process's rule set.

The following examples add rules to the configuration of an outbound server named xout.

Example 5–1 Adding Rules for the hr Schema, oe.orders Table, and oe.order_items Table to the Positive Rule Set

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name    => 'xout',
    table_names    => 'oe.orders, oe.order_items',
    schema_names   => 'hr',
    add            => TRUE,
    inclusion_rule => TRUE);
END;
/
```

Example 5–2 Adding Rules for the hr Schema to the Negative Rule Set

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name    => 'xout',
    table_names    => NULL,
    schema_names   => 'hr',
    add            => TRUE,
    inclusion_rule => FALSE);
END;
/
```

See Also: "ALTER_OUTBOUND Procedure" on page 8-14

Adding Subset Rules to an Outbound Server's Positive Rule Set This section describes adding subset rules to an outbound server's positive rule set using the ADD_SUBSET_OUTBOUND_RULES procedure in the DBMS_XSTREAM_ADM package. The ADD_SUBSET_OUTBOUND_RULES procedure only adds rules for DML changes to an outbound server's positive rule set. It does not add rules for DDL changes, and it does not add rules to a capture process's rule set.

To add subset rules to an outbound server's positive rule set:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the ADD_SUBSET_OUTBOUND_RULES procedure, and specify the following parameters:

   - server_name - Specify the name of the outbound server.

- table_name - Specify the table for which you want to capture and stream a subset of data.
- condition - Specify the subset condition, which is similar to the WHERE clause in a SQL statement, to stream changes to a subset of rows in the table.
- column_list - Specify the subset of columns to keep or discard, or specify NULL to keep all of the columns.
- keep - Specify TRUE to keep the columns listed in the column_list parameter, or specify FALSE to discard the columns in the column_list parameter.

When column_list is non-NULL and keep is set to TRUE, the procedure creates a keep columns declarative rule-based transformation for the columns listed in column_list.

When column_list is non-NULL and keep is set to FALSE, the procedure creates a delete column declarative rule-based transformation for each column listed in column_list.

3. If subset rules should also be added to the rule set of a capture process or propagation that streams row LCRs to the outbound server, then see *Oracle Streams Concepts and Administration* for information about adding rules to a rule set.

Example 5–3 Adding Rules That Stream Changes to a Subset of Rows in a Table

The following procedure creates rules that only evaluate to TRUE for row changes where the department_id value is 40 in the hr.employees table:

```
DECLARE
  cols DBMS_UTILITY.LNAME_ARRAY;
  BEGIN
    cols(1) := 'employee_id';
    cols(2) := 'first_name';
    cols(3) := 'last_name';
    cols(4) := 'email';
    cols(5) := 'phone_number';
    cols(6) := 'hire_date';
    cols(7) := 'job_id';
    cols(8) := 'salary';
    cols(9) := 'commission_pct';
    cols(10) := 'manager_id';
    cols(11) := 'department_id';
  DBMS_XSTREAM_ADM.ADD_SUBSET_OUTBOUND_RULES(
    server_name => 'xout',
    table_name  => 'hr.employees',
    condition   => 'department_id=40',
    column_list => cols);
END;
/
```

Example 5–4 Adding Rules That Stream Changes to a Subset of Rows and Columns in a Table

The following procedure creates rules that only evaluate to TRUE for row changes where the department_id value is 40 for the hr.employees table. The procedure also creates delete column declarative rule-based transformations for the salary and commission_pct columns.

```
BEGIN
  DBMS_XSTREAM_ADM.ADD_SUBSET_OUTBOUND_RULES(
    server_name => 'xout',
    table_name  => 'hr.employees',
    condition   => 'department_id=40',
    column_list => 'salary,commission_pct',
    keep        => FALSE);
END;
/
```

See Also:

- "ADD_SUBSET_OUTBOUND_RULES Procedure" on page 8-11
- *Oracle Streams Concepts and Administration* for information about declarative rule-based transformations

Removing Rules from an XStream Out Configuration

This section describes removing schema rules, table rules, and subset rules from an XStream Out configuration.

This section contains these topics:

- Removing Schema Rules and Table Rules From an XStream Out Configuration
- Removing Subset Rules from an Outbound Server's Positive Rule Set

Removing Schema Rules and Table Rules From an XStream Out Configuration This section describes removing schema rules and table rules from an XStream Out configuration using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package. The `ALTER_OUTBOUND` procedure removes rules for both DML and DDL changes.

When you follow the instructions in this section, the `ALTER_OUTBOUND` procedure always removes rules for the specified schemas and tables from one of the outbound server's rule sets. If the `DBMS_XSTREAM_ADM` package can manage the outbound server's capture process, then the `ALTER_OUTBOUND` procedure also removes rules for the specified schemas and tables from one of the rule sets used by this capture process.

To determine whether the `DBMS_XSTREAM_ADM` package can manage the outbound server's capture process, see "Checking Whether the DBMS_XSTREAM_ADM Package Can Manage a Capture Process" on page 5-2. If the `DBMS_XSTREAM_ADM` package cannot manage the outbound server's capture process, then the `ALTER_OUTBOUND` procedure removes rules from the outbound server's rule set only. In this case, if you must remove the rules for same schemas and tables from the capture process's rule set as well, then see *Oracle Streams Concepts and Administration* for instructions.

In addition, if the capture process is running on a different database than the outbound server, then remove the schema and table rules from the propagation that sends LCRs to the outbound server's database. See *Oracle Streams Concepts and Administration* for instructions.

To remove schema rules and table rules from an XStream Out configuration:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `ALTER_OUTBOUND` procedure, and specify the following parameters:

   - `server_name` - Specify the name of the outbound server.
   - `table_names` - Specify the tables for which to remove rules, or specify `NULL` to remove no table rules.

- schema_name - Specify the schemas for which to remove rules, or specify NULL to remove no schema rules.
- add - Specify FALSE so that the rules are removed. (Rules are added if you specify TRUE.)
- inclusion_rule - Specify TRUE to remove rules from the positive rule set of the outbound server, or specify FALSE to remove rules from the negative rule set of the outbound server. If the DBMS_XSTREAM_ADM package can manage the outbound server's capture process, then rules are also removed from this capture process's rule set.

The following examples remove rules from the configuration of an outbound server named xout.

Example 5–5 Removing Rules for the hr Schema, oe.orders Table, and oe.order_items Table from the Positive Rule Set

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name    => 'xout',
    table_names    => 'oe.orders, oe.order_items',
    schema_names   => 'hr',
    add            => FALSE,
    inclusion_rule => TRUE);
END;
/
```

Example 5–6 Removing Rules for the hr Schema from the Negative Rule Set

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name    => 'xout',
    table_names    => NULL,
    schema_names   => 'hr',
    add            => FALSE,
    inclusion_rule => FALSE);
END;
/
```

See Also: "ALTER_OUTBOUND Procedure" on page 8-14

Removing Subset Rules from an Outbound Server's Positive Rule Set This section describes removing subset rules from an outbound server's positive rule set using the REMOVE_SUBSET_OUTBOUND_RULES procedure in the DBMS_XSTREAM_ADM package. The REMOVE_SUBSET_OUTBOUND_RULES procedure only removes rules for DML changes. It does not remove rules for DDL changes, and it does not remove rules from a capture process's rule set.

To remove subset rules from an outbound server's positive rule set:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Determine the rule names for the subset rules by running the following query:

```
SELECT RULE_OWNER, SUBSETTING_OPERATION, RULE_NAME
   FROM DBA_XSTREAM_RULES
   WHERE SUBSETTING_OPERATION IS NOT NULL;
```

3. Run the `REMOVE_SUBSET_OUTBOUND_RULES` procedure, and specify the rules to remove from the list of rules displayed in Step 2.

   For example, assume that Step 2 returned the following results:

```
RULE_OWNER                     SUBSET RULE_NAME
------------------------------ ------ ------------------------------
XSTRMADMIN                     INSERT EMPLOYEES71
XSTRMADMIN                     UPDATE EMPLOYEES72
XSTRMADMIN                     DELETE EMPLOYEES73
```

Example 5–7 Removing Subset Rules From an Outbound Server's Positive Rule Set

To remove these rules from the positive rule set of the `xout` outbound server, run the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.REMOVE_SUBSET_OUTBOUND_RULES(
    server_name      => 'xout',
    insert_rule_name => 'xstrmadmin.employees71',
    update_rule_name => 'xstrmadmin.employees72',
    delete_rule_name => 'xstrmadmin.employees73');
END;
/
```

4. If subset rules should also be removed from the rule set of a capture process and propagation that streams row LCRs to the outbound server, then see *Oracle Streams Concepts and Administration* for information about removing rules.

   See Also: "REMOVE_SUBSET_OUTBOUND_RULES Procedure" on page 8-30

Changing the Connect User for an Outbound Server

A client application can connect to an outbound server as the connect user. This section describes changing the connect user for an outbound server using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

The connect user is the user who can attach to the outbound server to retrieve the LCR stream. The client application must attach to the outbound server as the connect user.

You can change the connect user when a client application must connect to an outbound server as a different user. Ensure that the connect user is granted the required privileges.

See Also: "CREATE_OUTBOUND Procedure" on page 8-21 for information about the privileges required by a connect user

To change the connect user for an outbound server:

1. Connect to the outbound server database as the XStream administrator.

   The XStream administrator must be granted the `DBA` role to change the connect user for an outbound server.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `ALTER_OUTBOUND` procedure, and specify the following parameters:

   - `server_name` - Specify the name of the outbound server.

- `connect_user` - Specify the new connect user.

Example 5–8 Changing the Connect User for an Outbound Server

To change the connect user to `hr` for an outbound server named `xout`, run the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name  => 'xout',
    connect_user => 'hr');
END;
/
```

See Also: "ALTER_OUTBOUND Procedure" on page 8-14

Changing the Capture User of the Capture Process for an Outbound Server

A capture user is the user in whose security domain a capture process captures changes from the redo log. This section describes changing the capture user for a capture process that captures changes for an outbound server using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

You can change the capture user when the capture process must capture changes in a different security domain. Ensure that the capture user is granted the required privileges. When you change the capture user, the `ALTER_OUTBOUND` procedure grants the new capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user.

Note: If Oracle Database Vault is installed, then the user who changes the capture user must be granted the `BECOME USER` system privilege. Granting this privilege to the user is not required if Oracle Database Vault is not installed. You can revoke the `BECOME USER` system privilege from the user after capture user is changed, if necessary.

See Also: "CREATE_OUTBOUND Procedure" on page 8-21 for information about the privileges required by a capture user

To change the capture user of the capture process for an outbound server:

1. Connect to the outbound server database as the XStream administrator.

   To change the capture user, the user who invokes the `ALTER_OUTBOUND` procedure must be granted `DBA` role. Only the `SYS` user can set the capture user to `SYS`.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Determine whether the `DBMS_XSTREAM_ADM` package can manage the capture process. See "Checking Whether the DBMS_XSTREAM_ADM Package Can Manage a Capture Process" on page 5-2.

   Based on the check, follow the appropriate instructions:

   - If the capture process can be managed using the `DBMS_XSTREAM_ADM` package, then proceed to Step 3.

- If the capture process cannot be managed using the `DBMS_XSTREAM_ADM` package, then follow the instructions in *Oracle Streams Concepts and Administration*.

3. Run the `ALTER_OUTBOUND` procedure, and specify the following parameters:
   - `server_name` - Specify the name of the outbound server.
   - `capture_user` - Specify the new capture user.

Example 5–9 Changing the Capture User of the Capture Process for an Outbound Server

To change the capture user to `hq_admin` for an outbound server named `xout`, run the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name  => 'xout',
    capture_user => 'hq_admin');
END;
/
```

See Also: "ALTER_OUTBOUND Procedure" on page 8-14

Changing the Start SCN or Start Time of the Capture Process for an Outbound Server

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

This section describes changing the start system change number (SCN) or start time for a capture process that captures changes for an outbound server using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

The start SCN is the SCN from which a capture process begins to capture changes. The start time is the time from which a capture process begins to capture changes. When you reset a start SCN or start time for a capture process, ensure that the required redo log files are available to the capture process.

Typically, you reset the start SCN or start time for a capture process if point-in-time recovery was performed on one of the destination databases that receive changes from the capture process.

This section contains these topics:

- Changing the Start SCN of the Capture Process for an Outbound Server
- Changing the Start Time of the Capture Process for an Outbound Server

Changing the Start SCN of the Capture Process for an Outbound Server

This section describes changing the start SCN of the capture process for an outbound server.

To change the start SCN for a capture process:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Check the first SCN of the capture process:

```
COLUMN CAPTURE_PROCESS HEADING 'Capture Process Name' FORMAT A30
COLUMN FIRST_SCN HEADING 'First SCN' FORMAT 99999999999999

SELECT CAPTURE_NAME, FIRST_SCN FROM DBA_CAPTURE;

CAPTURE_NAME                         First SCN
------------------------------ ---------------
CAP$_XOUT_1                             604426
```

When you reset the start SCN, the specified start SCN must be equal to or greater than the first SCN for the capture process.

3. Run the `ALTER_OUTBOUND` procedure, and specify the following parameters:
   - `server_name` - Specify the name of the outbound server.
   - `start_scn` - Specify the SCN from which the capture process begins to capture changes.

   If the capture process is enabled, then the `ALTER_OUTBOUND` procedure automatically stops and restarts the capture process when the `start_scn` parameter is non-`NULL`.

   If the capture process is disabled, then the `ALTER_OUTBOUND` procedure automatically starts the capture process when the `start_scn` parameter is non-`NULL`.

Example 5–10   Setting the Start SCN of the Capture Process for an Outbound Server

Run the following procedure to set the start SCN to `650000` for the capture process used by the `xout` outbound server:

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name => 'xout',
    start_scn    => 650000);
END;
/
```

See Also: "ALTER_OUTBOUND Procedure" on page 8-14

Changing the Start Time of the Capture Process for an Outbound Server

This section describes changing the start time of the capture process for an outbound server.

To change the start time for a capture process:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Check the time that corresponds with the first SCN of the capture process:

```
COLUMN CAPTURE_PROCESS HEADING 'Capture Process Name' FORMAT A30
COLUMN FIRST_SCN HEADING 'First SCN' FORMAT A40

SELECT CAPTURE_NAME, SCN_TO_TIMESTAMP(FIRST_SCN) FIRST_SCN FROM DBA_CAPTURE;

CAPTURE_NAME                   First SCN
------------------------------ ----------------------------------------
CAP$_XOUT_1                    05-MAY-10 08.11.17.000000000 AM
```

When you reset the start time, the specified start time must be greater than or equal to the time that corresponds with the first SCN for the capture process.

3. Run the `ALTER_OUTBOUND` procedure, and specify the following parameters:

   - `server_name` - Specify the name of the outbound server.
   - `start_time` - Specify the time from which the capture process begins to capture changes.

   If the capture process is enabled, then the `ALTER_OUTBOUND` procedure automatically stops and restarts the capture process when the `start_time` parameter is non-`NULL`.

   If the capture process is disabled, then the `ALTER_OUTBOUND` procedure automatically starts the capture process when the `start_time` parameter is non-`NULL`.

   The following examples set the `start_time` parameter for the capture process that captures changes for an outbound server named `xout`.

Example 5–11 Set the Start Time to a Specific Time

Run the following procedure to set the start time to `05-MAY-10 11.11.17 AM` for the capture process used by the `xout` outbound server:

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name => 'xout',
    start_time  => '05-MAY-10 11.11.17 AM');
END;
/
```

Example 5–12 Set the Start Time Using the NUMTODSINTERVAL SQL Function

Run the following procedure to set the start time to four hours earlier than the current time for the capture process used by the `xout` outbound server:

```
DECLARE
  ts  TIMESTAMP;
BEGIN
  ts := SYSTIMESTAMP - NUMTODSINTERVAL(4, 'HOUR');
  DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name => 'xout',
    start_time  => ts);
END;
/
```

See Also: "ALTER_OUTBOUND Procedure" on page 8-14

Dropping Components in an XStream Out Configuration

This section describes dropping an outbound server using the `DROP_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

This procedure always drops the specified outbound server. This procedure also drops the queue used by the outbound server if both of the following conditions are met:

- The queue was created by the `ADD_OUTBOUND` or `CREATE_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

- The outbound server is the only subscriber to the queue.

If either one of the preceding conditions is not met, then the DROP_OUTBOUND procedure only drops the outbound server. It does not drop the queue.

This procedure also drops the capture process for the outbound server if both of the following conditions are met:

- The procedure can drop the outbound server's queue.
- The DBMS_XSTREAM_ADM package can manage the outbound server's capture process. See "Checking Whether the DBMS_XSTREAM_ADM Package Can Manage a Capture Process" on page 5-2.

If the procedure can drop the queue but cannot manage the capture process, then it drops the queue without dropping the capture process.

To drop an outbound server:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the DROP_OUTBOUND procedure.

Example 5–13 Dropping an Outbound Server

To drop an outbound server named xout, run the following procedure:

```
exec DBMS_XSTREAM_ADM.DROP_OUTBOUND('xout');
```

See Also:

- "DROP_OUTBOUND Procedure" on page 8-26
- *Oracle Streams Concepts and Administration* for information about dropping a queue or a capture process

Managing XStream In

This section describes managing an XStream In inbound server configuration.

This section contains these topics:

- Changing the Apply User for an Inbound Server
- Managing Eager Errors Encountered by an Inbound Server
- Dropping Components in an XStream In Configuration

Note: With XStream In, an Oracle Streams apply process functions as an inbound server. Therefore, you can use the instructions for managing an apply process to manage an inbound server. See *Oracle Database 2 Day + Data Replication and Integration Guide* and *Oracle Streams Concepts and Administration*.

See Also:

- "XStream In" on page 2-9
- "Configuring XStream In" on page 4-24
- "Monitoring XStream In" on page 6-10

Changing the Apply User for an Inbound Server

An inbound server applies messages in the security domain of its apply user, and the client application must attach to the inbound server as the apply user. This section describes changing the apply user for an inbound server using the `ALTER_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

You can change the apply user when a client application must connect to an inbound server as a different user or when you want to apply changes using the privileges associated with a different user. Ensure that the apply user is granted the required privileges.

See Also: "CREATE_INBOUND Procedure" on page 8-19 for information about the privileges required by an apply user

To change the apply user for an inbound server:

1. Connect to the inbound server database as the XStream administrator.

   The XStream administrator must be granted the `DBA` role to change the apply user for an inbound server.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `ALTER_INBOUND` procedure, and specify the following parameters:

   - `server_name` - Specify the name of the inbound server.
   - `apply_user` - Specify the new apply user.

Example 5–14 Changing the Apply User for an Inbound Server

To change the apply user to `hr` for an inbound server named `xin`, run the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.ALTER_INBOUND(
    server_name => 'xin',
    apply_user  => 'hr');
END;
/
```

See Also:

- "ALTER_INBOUND Procedure" on page 8-13
- "Security Model" on page 8-4 for information about the security requirements for configuring and managing XStream
- *Oracle Streams Concepts and Administration*

Managing Eager Errors Encountered by an Inbound Server

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

An inbound server can encounter an eager error when it cannot access all of the LCRs in an error transaction. The `EAGER ERROR` error type typically means that an LCR raised an error while the inbound server was receiving and applying LCRs in a large transaction. If an error transaction is not an eager error transaction, then it is referred to as a normal error transaction.

Normal error transactions and eager error transactions must be managed differently. An inbound server moves a normal error transaction, including all of its LCRs, to the error queue, but an inbound server does not move an eager error transaction to the error queue.

The following statements apply to both normal error transactions and eager error transactions:

- The `ALL_APPLY_ERROR` and the `DBA_APPLY_ERROR` view contain information (metadata) about the error transaction.
- The inbound server does not apply the error transaction.

Table 5–1 explains the options for managing a normal error transaction.

Table 5–1 Options Available for Managing a Normal Error Transaction

Action	**Mechanisms**	**Description**

Delete the error transaction	`DBMS_APPLY_ADM.DELETE_ERROR`
	`DBMS_APPLY_ADM.DELETE_ALL_ERRORS`
	Oracle Enterprise Manager
		The error transaction is deleted from the error queue, and the metadata about the error transaction is deleted. An inbound server does not try to reexecute the transaction when the inbound server is restarted. The transaction is not applied.

Execute the error transaction	`DBMS_APPLY_ADM.EXECUTE_ERROR`
	`DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS`
	Oracle Enterprise Manager
		The error transaction in the error queue is executed. If there are no errors during execution, then the transaction is applied. If an LCR raises an error during execution, then the normal error transaction is moved back to the error queue.

Retain the error transaction	None. (The error transaction is retained automatically.)	The error transaction remains in the error queue even if the inbound server is restarted. The metadata about the error transaction is also retained. The transaction is not applied.

Table 5–2 explains the options for managing an eager error transaction.

Table 5–2 Options Available for Managing an Eager Error Transaction

Action	Mechanisms	Description

Delete error transaction	`DBMS_APPLY_ADM.DELETE_ERROR`
	`DBMS_APPLY_ADM.DELETE_ALL_ERRORS`
	Oracle Enterprise Manager
		The metadata about the eager error transaction is deleted. When the inbound server is restarted, it attempts to execute the transaction as an eager transaction. If the inbound server does not encounter an error during execution, then the transaction is applied successfully. If the inbound server encounters an error during execution, then the eager error transaction is recorded.

Retain error transaction	None. (The metadata about the error transaction is retained automatically.)	The metadata about the eager error transaction is retained. When the inbound server is restarted, it attempts to execute the transaction as a normal transaction. Specifically, the inbound server spills the transaction to disk and attempts to execute the transaction. If the inbound server does not encounter an error during execution, then the transaction is applied successfully. If the inbound server encounters an error during execution, then the transaction becomes a normal error transaction. In this case, the LCR that raised the error and all of the other LCRs in the transaction are moved to the error queue. After the normal error transaction is moved to the error queue, you must manage the error transaction as a normal error transaction (not an eager error transaction).

Note: If you attempt to execute an eager error transaction manually using the `DBMS_APPLY_ADM` package or Oracle Enterprise Manager, then the following error is raised:

```
ORA-26909: cannot reexecute an eager error
```

An eager error transaction cannot be executed manually. Instead, it is executed automatically when the inbound server is enabled.

To manage an eager error transaction encountered by an inbound server:

1. Connect to the inbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Query the `ERROR_TYPE` column in the `ALL_APPLY_ERROR` data dictionary view:

   ```
   SELECT APPLY_NAME, ERROR_TYPE FROM ALL_APPLY_ERROR;
   ```

   Follow the appropriate instructions based on the error type:

   - If the `ERROR_TYPE` column shows `EAGER ERROR`, then proceed to Step 3.

- If the ERROR_TYPE column shows NULL, then the apply error is not an eager error, and you cannot use the instructions in this section to manage it. Instead, use the instructions in *Oracle Database 2 Day + Data Replication and Integration Guide* or *Oracle Streams Concepts and Administration*.

3. Examine the error message raised by the LCR, and determine the cause of the error.

   See *Oracle Database 2 Day + Data Replication and Integration Guide* for information about checking for apply errors using Oracle Enterprise Manager.

   See *Oracle Streams Concepts and Administration* for information about checking for apply errors using the DBA_APPLY_ERROR data dictionary view.

4. If possible, determine how to avoid the error, and make any changes necessary to avoid the error.

   Oracle Streams Concepts and Administration contains information about common apply errors.

5. Either retain the error transaction or delete the error transaction:

   - Delete the error transaction only if you have corrected the problem. The inbound server reexecutes the transaction when it is enabled.
   - Retain the error transaction if you cannot correct the problem now or if you plan to reexecute it in the future.

   See Table 5–2 for more information about these choices.

   Caution: It might not be possible to recover a normal error transaction that is deleted. Before deleting the error transaction, ensure that the error type is EAGER ERROR.

   See *Oracle Database 2 Day + Data Replication and Integration Guide* for information about deleting an error transaction using Oracle Enterprise Manager.

   See *Oracle Streams Concepts and Administration* information about deleting an error transaction using the DBMS_APPLY_ADM package.

6. If the inbound server is disabled, then start the inbound server.

   Query the STATUS column in the ALL_APPLY_ERROR view to determine whether the inbound server is enabled or disabled.

   If the disable_on_error apply parameter is set to Y for the inbound server, then the inbound server becomes disabled when it encounters the error and remains disabled.

   If the disable_on_error apply parameter is set to N for the inbound server, then the inbound server stops and restarts automatically when it encounters the error.

   See Table 5–2 for information about how the inbound server handles the error transaction based on your choice in Step 5.

   See *Oracle Database 2 Day + Data Replication and Integration Guide* for information about starting an apply process (or inbound server) using Oracle Enterprise Manager.

   See *Oracle Streams Concepts and Administration* for information about starting an apply process (or inbound server) using the DBMS_APPLY_ADM package.

Note: If you have both purchased a license for the Oracle GoldenGate product and have enabled the XStream optimizations for Oracle Streams by running the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure, then an apply process in an Oracle Streams configuration can encounter errors of the `EAGER ERROR` type. Use the instructions in this section to manage eager apply process errors. When the XStream optimizations for Oracle Streams are not enabled, apply processes cannot encounter eager errors.

See Also:

- "ALL_APPLY_ERROR" on page 12-3
- "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" on page 8-27

Dropping Components in an XStream In Configuration

This section describes dropping an inbound server using the `DROP_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

This procedure always drops the specified inbound server. This procedure also drops the queue for the inbound server if both of the following conditions are met:

- One call to the `CREATE_INBOUND` procedure created the inbound server and the queue.
- The inbound server is the only subscriber to the queue.

If either one of the preceding conditions is not met, then the `DROP_INBOUND` procedure only drops the inbound server. It does not drop the queue.

To drop an inbound server:

1. Connect to the inbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `DROP_INBOUND` procedure.

Example 5–15 Dropping an Inbound Server

To drop an inbound server named `xin`, run the following procedure:

```
exec DBMS_XSTREAM_ADM.DROP_INBOUND('xin');
```

See Also: "DROP_INBOUND Procedure" on page 8-25

6 Monitoring XStream

This chapter provides instructions for monitoring XStream.

This chapter contains these topics:

See Also:

- Chapter 2, "XStream Concepts"
- Chapter 3, "XStream Use Cases"
- Chapter 4, "Configuring XStream"
- Chapter 5, "Managing XStream"
- Chapter 7, "Troubleshooting XStream"

About Monitoring XStream

This chapter describes monitoring an XStream Out configuration and an XStream In configuration. This chapter provides instructions for querying data dictionary views related to XStream. The queries provide information about XStream components and statistics related to XStream.

The main interface for monitoring XStream database components is SQL*Plus, although you can monitor some aspects of an XStream configuring using Oracle Enterprise Manager. For example, you can view information about capture processes, outbound servers, inbound servers, and rules in Enterprise Manager. Outbound servers and inbound servers appear as apply processes in Enterprise Manager.

This chapter also describes using the Oracle Streams Performance Advisor to monitor an XStream configuration. The Oracle Streams Performance Advisor consists of the `DBMS_STREAMS_ADVISOR_ADM` package and a collection of data dictionary views. The Oracle Streams Performance Advisor enables you to monitor the topology and performance of an XStream environment.

Monitoring Session Information About XStream Components

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

The query in this section displays the following session information about each XStream component in a database:

- The XStream component name
- The session identifier
- The serial number
- The operating system process identification number
- The XStream process number

This query is especially useful for determining the session information for specific XStream components when there are multiple XStream Out or XStream In components configured in a database.

To display this information for each XStream component in a database:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN ACTION HEADING 'XStream Component' FORMAT A30
COLUMN SID HEADING 'Session ID' FORMAT 99999
COLUMN SERIAL# HEADING 'Session|Serial|Number' FORMAT 99999999
COLUMN PROCESS HEADING 'Operating System|Process Number' FORMAT A17
COLUMN PROCESS_NAME HEADING 'XStream|Process|Number' FORMAT A7

SELECT /*+PARAM('_module_action_old_length',0)*/ ACTION,
       SID,
       SERIAL#,
       PROCESS,
       SUBSTR(PROGRAM,INSTR(PROGRAM,'(')+1,4) PROCESS_NAME
  FROM V$SESSION
  WHERE MODULE ='XStream';
```

Your output for an XStream Out configuration looks similar to the following:

```
                                         Session                   XStream
                                          Serial Operating System  Process
XStream Component              Session ID  Number Process Number    Number
------------------------------ ---------- --------- ----------------- -------
XOUT - Apply Server                    35        33 16386             TNS
XOUT - Apply Coordinator               41         1 14093             AP01
XOUT - Apply Reader                    43         1 14095             AS01
XOUT - Apply Server                    45         1 14097             AS02
XOUT - Propagation Send/Rcv            47        55 16401             CS01
CAP$_XOUT_1 - Capture                  48         7 16399             CP01
```

The row that shows `TNS` for the XStream process number contains information about the session for the XStream client application that is attached to the outbound server.

Your output for an XStream In configuration looks similar to the following:

```
                                                 Session                   XStream
                                                  Serial Operating System  Process
XStream Component              Session ID         Number Process Number    Number
------------------------------ ---------- --------- ----------------- -------
XIN - Propagation Receiver             32        21 16386             TNS
XIN - Apply Coordinator                38        23 16414             AP01
XIN - Apply Reader                     40         3 16418             AS01
XIN - Apply Server                     42         1 16420             AS02
XIN - Apply Server                     44         1 16422             AS03
XIN - Apply Server                     46         1 16424             AS04
XIN - Apply Server                     48         1 16426             AS05
```

The row that shows `TNS` for the XStream process number contains information about the session for the XStream client application that is attached to the inbound server.

See Also: *Oracle Database Reference* for more information about the `V$SESSION` view

Monitoring XStream Out

This section provides sample queries that you can use to monitor XStream Out.

This section contains these topics:

- Displaying General Information About an Outbound Server
- Displaying Status and Error Information for an Outbound Server
- Displaying Information About an Outbound Server's Current Transaction
- Displaying Statistics for an Outbound Server
- Displaying the Processed Low Position for an Outbound Server
- Determining the Process Information for an Outbound Server

With XStream Out, an Oracle Streams apply process functions as an outbound server. Therefore, you can also use the data dictionary views for apply processes to monitor outbound servers. In addition, an XStream Out environment includes capture processes and queues, and might include other components, such as propagations, rules, and rule-based transformations.

See Also: *Oracle Streams Concepts and Administration*

Displaying General Information About an Outbound Server

You can display the following information for an outbound server by running the query in this section:

- The outbound server name
- The name of the connect user for the outbound server

  The connect user is the user who can attach to the outbound server to retrieve the logical change record (LCR) stream. The client application must attach to the outbound server as the specified connect user.
- The name of the capture user for the capture process that captures changes for the outbound server to process
- The name of the capture process that captures changes for the outbound server to process
- The name of the source database for the captured changes

- The owner of the queue used by the outbound server
- The name of the queue used by the outbound server

The DBA_XSTREAM_OUTBOUND view contains information about the capture user, the capture process, and the source database in either of the following cases:

- The outbound server was created using the CREATE_OUTBOUND procedure in the DBMS_XSTREAM_ADM package.
- The outbound server was created using the ADD_OUTBOUND procedure in the DBMS_XSTREAM_ADM package, and the capture process for the outbound server runs on the same database as the outbound server.

If the outbound server was created using the ADD_OUTBOUND procedure, and the capture process for the outbound server is on a different database, then the DBA_XSTREAM_OUTBOUND view does not contain information about the capture user, the capture process, or the source database.

To display this general information about an outbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Outbound|Server|Name' FORMAT A10
COLUMN CONNECT_USER HEADING 'Connect|User' FORMAT A10
COLUMN CAPTURE_USER HEADING 'Capture|User' FORMAT A10
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A11
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A11
COLUMN QUEUE_OWNER HEADING 'Queue|Owner' FORMAT A10
COLUMN QUEUE_NAME HEADING 'Queue|Name' FORMAT A10

SELECT SERVER_NAME,
       CONNECT_USER,
       CAPTURE_USER,
       CAPTURE_NAME,
       SOURCE_DATABASE,
       QUEUE_OWNER,
       QUEUE_NAME
  FROM DBA_XSTREAM_OUTBOUND;
```

Your output looks similar to the following:

```
Outbound                         Capture
Server     Connect    Capture    Process     Source      Queue      Queue
Name       User       User       Name        Database    Owner      Name
---------- ---------- ---------- ----------- ----------- ---------- ----------
XOUT       XSTRMADMIN XSTRMADMIN CAP$_XOUT_1 DB.EXAMPLE. XSTRMADMIN Q$_XOUT_2
                                             COM
```

See Also:

- "ALL_XSTREAM_OUTBOUND" on page 12-8
- *Oracle Streams Concepts and Administration*

Displaying Status and Error Information for an Outbound Server

You can monitor an outbound server using the same queries as you use to monitor an Oracle Streams apply process. See *Oracle Streams Concepts and Administration* for instructions.

The `ALL_APPLY` and `DBA_APPLY` views show `XStream Out` in the `PURPOSE` column for an apply process that is functioning as an outbound server.

To display detailed information about an outbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

   ```
   COLUMN APPLY_NAME HEADING 'Apply Name' FORMAT A10
   COLUMN STATUS HEADING 'Status' FORMAT A8
   COLUMN ERROR_NUMBER HEADING 'Error Number' FORMAT 9999999
   COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A40

   SELECT APPLY_NAME,
          STATUS,
          ERROR_NUMBER,
          ERROR_MESSAGE
     FROM DBA_APPLY
     WHERE PURPOSE = 'XStream Out';
   ```

Your output looks similar to the following:

```
Apply Name Status   Error Number Error Message
---------- -------- ------------ ----------------------------------------
XOUT       ENABLED
```

This output shows that `XOUT` is an apply process that is functioning as an outbound server. Use the instructions in *Oracle Streams Concepts and Administration* to display detailed information about the outbound server.

See Also: "ALL_APPLY" on page 12-1

Displaying Information About an Outbound Server's Current Transaction

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

The `V$XSTREAM_OUTBOUND_SERVER` view contains the following information about the transaction currently being processed by an XStream outbound server:

- The name of the outbound server
- The transaction ID of the transaction currently being processed
- Commit system change number (SCN) of the transaction currently being processed
- Commit position of the transaction currently being processed
- The position of the last LCR sent to the XStream client application
- The message number of the current LCR being processed by the outbound server

Run this query to determine how many LCRs an outbound server has processed in a specific transaction. You can query the `TOTAL_MESSAGE_COUNT` column in the `V$XSTREAM_TRANSACTION` view to determine the total number of LCRs in a transaction.

To display information about an outbound server's current transaction:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Outbound|Server|Name' FORMAT A10
COLUMN 'Transaction ID' HEADING 'Transaction|ID' FORMAT A11
COLUMN COMMITSCN HEADING 'Commit SCN' FORMAT 9999999999999
COLUMN COMMIT_POSITION HEADING 'Commit Position' FORMAT A15
COLUMN LAST_SENT_POSITION HEADING 'Last Sent|Position' FORMAT A15
COLUMN MESSAGE_SEQUENCE HEADING 'Message|Number' FORMAT 999999999

SELECT SERVER_NAME,
       XIDUSN ||'.'||
       XIDSLT ||'.'||
       XIDSQN "Transaction ID",
       COMMITSCN,
       COMMIT_POSITION,
       LAST_SENT_POSITION,
       MESSAGE_SEQUENCE
  FROM V$XSTREAM_OUTBOUND_SERVER;
```

Your output looks similar to the following:

```
Outbound
Server     Transaction                                Last Sent          Message
Name       ID              Commit SCN Commit Position Position            Number
---------- ----------- -------------- --------------- --------------- ----------
XOUT       17.23.59            645856 00000009DAE0000 00000009DAE0000          4
                                      000010000000100 000010000000100
                                      000009DAE000000 000009DAE000000
                                      0010000000101   0010000000101
```

Note: The `COMMITSCN` and `COMMIT_POSITION` values are populated only if the `COMMITTED_DATA_ONLY` value is `YES` in `V$XSTREAM_OUTBOUND_SERVER`.

See Also:

- "V$XSTREAM_OUTBOUND_SERVER" on page 13-10
- "V$XSTREAM_TRANSACTION" on page 13-13

Displaying Statistics for an Outbound Server

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

The V$XSTREAM_OUTBOUND_SERVER view contains the following statistics about the database changes processed by an XStream outbound server:

- The name of the outbound server
- The number of transactions sent from the outbound server to the XStream client application since the last time the client application attached to the outbound server
- The number of LCRs sent from the outbound server to the XStream client application since the last time the client application attached to the outbound server
- The number of megabytes sent from the outbound server to the XStream client application since the last time the client application attached to the outbound server
- The amount of time the outbound server spent sending LCRs to the XStream client application since the last time the client application attached to the outbound server
- The message number of the last LCR sent by the outbound server to the XStream client application
- Creation time at the source database of the last LCR sent by the outbound server to the client application

To display statistics for an outbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Outbound|Server|Name' FORMAT A8
COLUMN TOTAL_TRANSACTIONS_SENT HEADING 'Total|Trans|Sent' FORMAT 9999999
COLUMN TOTAL_MESSAGES_SENT HEADING 'Total|LCRs|Sent' FORMAT 9999999999
COLUMN BYTES_SENT HEADING 'Total|MB|Sent' FORMAT 99999999999999
COLUMN ELAPSED_SEND_TIME HEADING 'Time|Sending|LCRs|(in seconds)' FORMAT
99999999
COLUMN LAST_SENT_MESSAGE_NUMBER HEADING 'Last|Sent|Message|Number' FORMAT
99999999
COLUMN LAST_SENT_MESSAGE_CREATE_TIME HEADING 'Last|Sent|Message|Creation|Time'
FORMAT A10

SELECT SERVER_NAME,
       TOTAL_TRANSACTIONS_SENT,
       TOTAL_MESSAGES_SENT,
       (BYTES_SENT/1024)/1024 BYTES_SENT,
       (ELAPSED_SEND_TIME/100) ELAPSED_SEND_TIME,
       LAST_SENT_MESSAGE_NUMBER,
       LAST_SENT_MESSAGE_CREATE_TIME
  FROM V$XSTREAM_OUTBOUND_SERVER;
```

Your output looks similar to the following:

```
                                                                     Last
                                                         Time      Last Sent
Outbound    Total       Total           Total       Sending       Sent Message
Server      Trans        LCRs              MB          LCRs    Message Creation
Name         Sent        Sent            Sent  (in seconds)     Number Time
-------- -------- ----------- --------------- ------------ --------- ----------
```

```
XOUT         2000      216000             56          291   9381070 4-AUG-10
                                                                    11:03 A.M.
```

Note: The `TOTAL_TRANSACTIONS_SENT` value is populated only if the `COMMITTED_DATA_ONLY` value is `YES` in `V$XSTREAM_OUTBOUND_SERVER`.

See Also: "V$XSTREAM_OUTBOUND_SERVER" on page 13-10

Displaying the Processed Low Position for an Outbound Server

For an outbound server, the processed low position is the position below which all transactions have been committed and logged by the client application. The processed low position is important when the outbound server or the client application is restarted.

You can display the following information about the processed low position for an outbound server by running the query in this section:

- The outbound server name
- The name of the source database for the captured changes
- The processed low position, which indicates the low watermark position processed by the client application
- The time when the processed low position was last updated by the outbound server

To display the processed low position for an outbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Outbound|Server|Name' FORMAT A10
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A20
COLUMN PROCESSED_LOW_POSITION HEADING 'Processed|Low LCR|Position' FORMAT A30
COLUMN PROCESSED_LOW_TIME HEADING 'Processed|Low|Time' FORMAT A9

SELECT SERVER_NAME,
       SOURCE_DATABASE,
       PROCESSED_LOW_POSITION,
       TO_CHAR(PROCESSED_LOW_TIME,'HH24:MI:SS MM/DD/YY') PROCESSED_LOW_TIME
FROM DBA_XSTREAM_OUTBOUND_PROGRESS;
```

Your output looks similar to the following:

```
Outbound                        Processed                      Processed
Server     Source               Low LCR                        Low
Name       Database             Position                       Time
---------- -------------------- ------------------------------ ---------
XOUT       DB.EXAMPLE.COM       00000008F17A000000000000000000 13:39:01
                                000008F17A000000000000000001   07/15/09
```

See Also:

- "ALL_XSTREAM_OUTBOUND_PROGRESS" on page 12-9
- "The Processed Low Position and Restartability for XStream Out" on page 2-12

Determining the Process Information for an Outbound Server

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

An outbound server is an Oracle background process. This background process runs only when an XStream client application attaches to the outbound server. The `V$XSTREAM_OUTBOUND_SERVER` view contains information about this background process.

You can display the following information for an outbound server by running the query in this section:

- The outbound server name
- The session ID of the outbound server's session
- The serial number of the outbound server's session
- The process identification number of the operating-system process that sends LCRs to the client application

To display the process information for an outbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Outbound Server Name' FORMAT A20
COLUMN SID HEADING 'Session ID' FORMAT 9999999999
COLUMN SERIAL# HEADING 'Serial Number' FORMAT 9999999999
COLUMN SPID HEADING 'Operating-System Process' FORMAT A25

SELECT SERVER_NAME,
       SID,
       SERIAL#,
       SPID
  FROM V$XSTREAM_OUTBOUND_SERVER;
```

Your output looks similar to the following:

```
Outbound Server Name  Session ID Serial Number Operating-System Process
-------------------- ----------- ------------- -------------------------
XOUT                          53           406 25783
```

Note: The `V$STREAMS_APPLY_SERVER` view provides additional information about the outbound server process, and information about the apply server background processes used by the outbound server.

See Also:

- "V$XSTREAM_OUTBOUND_SERVER" on page 13-10
- "V$STREAMS_APPLY_SERVER" on page 13-5

Monitoring XStream In

This section provides sample queries that you can use to monitor XStream In.

This section contains these topics:

- Displaying General Information About an Inbound Server
- Displaying the Status and Error Information for an Inbound Server
- Displaying the Position Information for an Inbound Server

With XStream In, an Oracle Streams apply process functions as an inbound server. Therefore, you can also use the data dictionary views for apply processes to monitor inbound servers.

See Also: *Oracle Streams Concepts and Administration*

Displaying General Information About an Inbound Server

You can display the following information for an inbound server by running the query in this section:

- The inbound server name
- The owner of the queue used by the inbound server
- The name of the queue used by the inbound server
- The apply user for the inbound server

To display general information about an inbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Inbound Server Name' FORMAT A20
COLUMN QUEUE_OWNER HEADING 'Queue Owner' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Queue Name' FORMAT A15
COLUMN APPLY_USER HEADING 'Apply User' FORMAT A15

SELECT SERVER_NAME,
       QUEUE_OWNER,
       QUEUE_NAME,
       APPLY_USER
  FROM DBA_XSTREAM_INBOUND;
```

Your output looks similar to the following:

```
Inbound Server Name  Queue Owner     Queue Name      Apply User
-------------------- --------------- --------------- ---------------
XIN                  XSTRMADMIN      XQUEUE          XSTRMADMIN
```

See Also: "ALL_XSTREAM_INBOUND" on page 12-6

Displaying the Status and Error Information for an Inbound Server

You can monitor an inbound server using the same queries that you use to monitor an Oracle Streams apply process. See *Oracle Streams Concepts and Administration* for instructions.

The `ALL_APPLY` and `DBA_APPLY` views show `XStream In` in the `PURPOSE` column for an apply process that is functioning as an inbound server.

To display the status of an inbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

   ```
   COLUMN APPLY_NAME HEADING 'Apply Name' FORMAT A10
   COLUMN STATUS HEADING 'Status' FORMAT A8
   COLUMN ERROR_NUMBER HEADING 'Error Number' FORMAT 9999999
   COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A40

   SELECT APPLY_NAME,
          STATUS,
          ERROR_NUMBER,
          ERROR_MESSAGE
     FROM DBA_APPLY
     WHERE PURPOSE = 'XStream In';
   ```

Your output looks similar to the following:

```
Apply Name Status   Error Number Error Message
---------- -------- ------------ ----------------------------------------
XIN        ENABLED
```

This output shows that `XIN` is an apply process that is functioning as an inbound server. Use the instructions in *Oracle Streams Concepts and Administration* to display detailed information about the inbound server.

See Also: "ALL_APPLY" on page 12-1

Displaying the Position Information for an Inbound Server

For an inbound server, you can view position information by querying the `DBA_XSTREAM_INBOUND_PROGRESS` view. Specifically, you can display the following position information by running the query in this section:

- The inbound server name
- The applied low position for the inbound server
- The spill position for the inbound server
- The applied high position for the inbound server
- The processed low position for the inbound server

To display the position information for an inbound server:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN SERVER_NAME HEADING 'Inbound|Server|Name' FORMAT A10
COLUMN APPLIED_LOW_POSITION HEADING 'Applied Low|Position' FORMAT A15
COLUMN SPILL_POSITION HEADING 'Spill Position' FORMAT A15
COLUMN APPLIED_HIGH_POSITION HEADING 'Applied High|Position' FORMAT A15
COLUMN PROCESSED_LOW_POSITION HEADING 'Processed Low|Position' FORMAT A15

SELECT SERVER_NAME,
       APPLIED_LOW_POSITION,
       SPILL_POSITION,
       APPLIED_HIGH_POSITION,
       PROCESSED_LOW_POSITION
  FROM DBA_XSTREAM_INBOUND_PROGRESS;
```

Your output looks similar to the following:

```
Inbound
Server     Applied Low                     Applied High    Processed Low
Name       Position        Spill Position  Position        Position
---------- --------------- --------------- --------------- ---------------
XIN        C10A            C11D            C10A            C11D
```

The values of the positions shown in the output were set by the client application that attaches to the inbound server. However, the inbound server determines which values are the current applied low position, spill position, applied high position, and processed low position.

See Also:

- "ALL_XSTREAM_INBOUND_PROGRESS" on page 12-7
- "Position of LCRs and XStream In" on page 2-13

Monitoring XStream Rules

The ALL_XSTREAM_RULES and DBA_XSTREAM_RULES views contain information about the rules used by outbound servers and inbound servers. If an outbound server was created using the CREATE_OUTBOUND procedure in the DBMS_XSTREAM_ADM package, then these views also contain information about the rules used by the capture process that sends changes to the outbound server. However, if an outbound server was created using the ADD_OUTBOUND procedure, then these views do not contain information about the capture process rules. Also, these views do not contain information about the rules used by any propagation in the stream from a capture process to an outbound server.

To display information about the rules used by XStream components:

1. Connect to the database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN STREAMS_NAME HEADING 'Oracle|Streams|Name' FORMAT A12
COLUMN STREAMS_TYPE HEADING 'Oracle|Streams|Type' FORMAT A11
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A10
COLUMN RULE_SET_TYPE HEADING 'Rule Set|Type' FORMAT A8
COLUMN STREAMS_RULE_TYPE HEADING 'Oracle|Streams|Rule|Level' FORMAT A7
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A11
COLUMN RULE_TYPE HEADING 'Rule|Type' FORMAT A4
```

```
SELECT STREAMS_NAME,
       STREAMS_TYPE,
       RULE_NAME,
       RULE_SET_TYPE,
       STREAMS_RULE_TYPE,
       SCHEMA_NAME,
       OBJECT_NAME,
       RULE_TYPE
  FROM DBA_XSTREAM_RULES;
```

Your output looks similar to the following:

```
Oracle       Oracle                          Streams
Streams      Streams     Rule       Rule Set Rule    Schema Object      Rule
Name         Type        Name       Type     Level   Name   Name        Type
------------ ----------- ---------- -------- ------- ------ ----------- ----
CAP$_XOUT_49 CAPTURE     DB52       POSITIVE GLOBAL                     DML
CAP$_XOUT_49 CAPTURE     DB53       POSITIVE GLOBAL                     DDL
XOUT         APPLY       DB55       POSITIVE GLOBAL                     DML
XOUT         APPLY       DB56       POSITIVE GLOBAL                     DDL
```

Notice that the `STREAMS_TYPE` is `APPLY` even though the rules are in the positive rule set for the outbound server `xout`. You can determine the purpose of an apply process by querying the `PURPOSE` column in the `DBA_APPLY` view.

To view information about the rules used by all components, including capture processes, propagations, apply processes, outbound servers, and inbound servers, you can query the `ALL_STREAMS_RULES` and `DBA_STREAMS_RULES` views. See *Oracle Streams Concepts and Administration* for sample queries that enable you to monitor rules.

See Also: "ALL_XSTREAM_RULES" on page 12-10

XStream and the Oracle Streams Performance Advisor

The Oracle Streams Performance Advisor consists of the `DBMS_STREAMS_ADVISOR_ADM` PL/SQL package and a collection of data dictionary views. The Oracle Streams Performance Advisor enables you to monitor the topology and performance of an XStream environment. The XStream topology includes information about the components in an XStream environment, the links between the components, and the way information flows from capture to consumption. The Oracle Streams Performance Advisor also provides information about how Oracle Streams components are performing.

Apply processes function as XStream outbound servers and inbound servers. In general, the Oracle Streams Performance Advisor works the same way for an Oracle Streams environment with apply processes and an XStream environment with outbound servers or inbound servers. This section describes important considerations about using the Oracle Streams Performance Advisor in an XStream environment.

This section contains these topics:

- XStream Components
- Topology and Stream Paths
- XStream and Component-Level Statistics
- The UTL_SPADV Package

See Also: *Oracle Streams Concepts and Administration* for detailed information about using the Oracle Streams Performance Advisor

XStream Components

The Oracle Streams Performance Advisor tracks the following types of components in an XStream environment:

- `QUEUE`
- `CAPTURE`
- `PROPAGATION SENDER`
- `PROPAGATION RECEIVER`
- `APPLY`

The preceding types are the same in an Oracle Streams environment and an XStream environment, except for `APPLY`. The `APPLY` component type can be an XStream outbound server or inbound server.

The following subcomponent types are possible for apply processes, outbound servers, and inbound servers:

- `PROPAGATION SENDER+RECEIVER` for sending LCRs from a capture process directly to an apply process or outbound server in a combined capture and apply optimization
- `APPLY READER` for a reader server
- `APPLY COORDINATOR` for a coordinator process
- `APPLY SERVER` for an apply server

In addition, the Oracle Streams Performance Advisor identifies a bottleneck component as the busiest component or the component with the least amount of idle time. In an XStream configuration, the XStream client application might be the bottleneck when `EXTERNAL` appears in the `ACTION_NAME` column of the `DBA_STREAMS_TP_PATH_BOTTLENECK` view.

Topology and Stream Paths

In the Oracle Streams topology, a **stream path** is a flow of messages from a source to a destination. A stream path begins where a capture process, synchronous capture, or application enqueues messages into a queue. A stream path ends where an apply process, outbound server, or inbound server dequeues the messages. The stream path might flow through multiple queues and propagations before it reaches an apply process, outbound server, or inbound server. Therefore, a single stream path can consist of multiple source/destination component pairs before it reaches last component.

The Oracle Streams topology only gathers information about a stream path if the stream path ends with an apply process, an outbound server, or an inbound server. The Oracle Streams topology does not track stream paths that end when a messaging client or an application that dequeues messages.

XStream and Component-Level Statistics

The Oracle Streams Performance Advisor tracks the following component-level statistics:

- The `MESSAGE APPLY RATE` is the average number of messages applied each second by the apply process, outbound server, or inbound server.
- The `TRANSACTION APPLY RATE` is the average number of transactions applied by the apply process, outbound server, or inbound server each second. Transactions typically include multiple messages.

An LCR can be applied in one of the following ways:

- An apply process or inbound server makes the change encapsulated in the LCR to a database object.
- An apply process or inbound server passes the LCR to an apply handler.
- If the LCR raises an error, then an apply process or inbound server sends the LCR to the error queue.
- An outbound server passes the LCR to an XStream client application. If the LCR raises an error, then the outbound server also reports the error to the client application.

Also, the Oracle Streams Performance Advisor tracks the `LATENCY` component-level statistics. `LATENCY` is defined in the following ways:

- For apply processes, the `LATENCY` is the amount of time between when the message was created at a source database and when the message was applied by the apply process at the destination database.
- For outbound servers, the `LATENCY` is amount of time between when the message was created at a source database and when the message was sent to the XStream client application.
- For inbound servers, the `LATENCY` is amount of time between when the message was created by the XStream client application and when the message was applied by the apply process.

When a capture process creates an LCR, the message creation time is the time when the redo entry for the database change was recorded. When an XStream client application creates an LCR, the message creation time is the time when the LCR was constructed.

See Also: *Oracle Streams Concepts and Administration* for more information about component-level statistics

The UTL_SPADV Package

The `UTL_SPADV` package provides subprograms to collect and analyze statistics for the XStream components in a distributed database environment. The package uses the Oracle Streams Performance Advisor to gather statistics, and the output is formatted so that it can be imported into a spreadsheet easily and analyzed.

The `UTL_SPADV` package works the same way for an Oracle Streams environment with apply processes and an XStream environment with outbound servers or inbound servers. However, there are some differences in the output for the `SHOW_STATS` procedure. This section describes the differences between the output for apply processes and the output for XStream outbound servers and inbound servers.

Note: The rest of this section assumes that you are familiar with the UTL_SPADV package and the SHOW_STATS output for apply processes. See *Oracle Streams Concepts and Administration* and *Oracle Database PL/SQL Packages and Types Reference* for detailed information about using the UTL_SPADV package.

The following sections describe the output for the SHOW_STATS procedure for outbound servers and inbound servers:

- Sample Output When an Outbound Server Is the Last Component in a Path
- Sample Output When an Inbound Server Is the Last Component in a Path

Sample Output When an Outbound Server Is the Last Component in a Path

The following is sample output for when an outbound server is the last component in a path:

```
OUTPUT
PATH 1 RUN_ID 2 RUN_TIME 2009-MAY-15 12:20:55 CCA Y
|<C> CAP$_XOUT_1 2733 2730 3392 LMR 8.3% 91.7% 0% "" LMP (1) 8.3% 91.7% 0% ""
LMB 8.3% 91.7% 0% "" CAP 8.3% 91.7% 0% "" |<Q> "XSTRMADMIN"."Q$_XOUT_2" 2730 0.01
4109 |<A> XOUT 2329 2.73 0 -1 PS+PR 8.3% 91.7% 0% "" APR 8.3% 91.7% 0% "" APC
100% 0% 0% "" APS (1) 8.3% 83.3% 8.3% "" |<B> "EXTERNAL"
.
.
.
```

Note: This output is for illustrative purposes only. Actual performance characteristics vary depending on individual configurations and conditions.

In this output, the A component is the outbound server XOUT. The output for when an outbound server is the last component in a path is similar to the output for when an apply process is the last component in a path. However, the apply server (APS) is not the last component because the outbound server connects to a client application. Statistics are not collected for the client application.

In an XStream Out configuration, the output can indicate flow control for the network because "SQL*Net more data to client" for an apply server is considered as a flow control event. If the output indicates flow control for an apply server, then either the network or the client application is considered the bottleneck component. In the previous output, EXTERNAL indicates that either the network or the client application is the bottleneck.

Other than these differences, you can interpret the statistics in the same way that you would for a path that ends with an apply process. Use the legend and the abbreviations to determine the statistics in the output.

Sample Output When an Inbound Server Is the Last Component in a Path

The following is sample output for when an inbound server is the last component in a path:

```
OUTPUT
PATH 1 RUN_ID 2 RUN_TIME 2009-MAY-16 10:11:38 CCA N
|<PR> "clientcap"=> 75% 0% 8.3% "CPU + Wait for CPU" |<Q> "XSTRMADMIN"."QUEUE2"  467 0.01 1
|<A> XIN 476 4.71 0 APR 100% 0% 0% "" APC 100% 0% 0% "" APS (4) 366.7% 0% 33.3% "CPU + Wait for CPU"
```

```
|<B> "EXTERNAL"
.
.
.
```

Note: This output is for illustrative purposes only. Actual performance characteristics vary depending on individual configurations and conditions.

In this output, the `A` component is the inbound server `XIN`. When an inbound server is the last component in a path, the XStream client application connects to the inbound server, and the inbound server applies the changes in the LCRs. The client application is not shown in the output.

The propagation receiver receives the LCRs from the client application. So, the propagation receiver is the first component shown in the output. In the previous sample output, the propagation receiver is named `clientcap`. In this case, `clientcap` is the source name given by the client application when it attaches to the inbound server.

If the propagation receiver is idle for a significant percentage of time, then either the network or the client application is considered a bottleneck component. In the previous output, `EXTERNAL` indicates that either the network or the client application is the bottleneck.

Other than these differences, you can interpret the statistics in the same way that you would for a path that ends with an apply process. Use the legend and the abbreviations to determine the statistics in the output.

7 Troubleshooting XStream

This chapter describes common problems you might encounter while using XStream and explains how to solve them.

This chapter contains the following topics:

- Diagnosing Problems with XStream
- Problems and Solutions for XStream
- How to Get More Help with XStream

See Also:

- Chapter 2, "XStream Concepts"
- Chapter 3, "XStream Use Cases"
- Chapter 4, "Configuring XStream"
- Chapter 5, "Managing XStream"
- Chapter 6, "Monitoring XStream"

Diagnosing Problems with XStream

With XStream, an Oracle Streams apply process can function as an outbound server or an inbound server. An XStream configuration can also include other components, such as capture processes, queues, propagations, rules, and rule-based transformations.

To diagnose problems with XStream, you can use many of the same techniques that are used to diagnose problems with Oracle Streams components. These techniques include the following:

- Viewing Oracle Streams alerts
- Using the Streams configuration report and health check script
- Handling performance problems because of an unavailable destination
- Checking the trace files and alert log for problems

See *Oracle Streams Concepts and Administration* for detailed information about these topics.

Problems and Solutions for XStream

In general, you can troubleshoot XStream outbound servers and inbound servers in the same way that you troubleshoot Oracle Streams apply processes. In addition, an XStream Out environment includes capture processes and queues, and might include other components, such as propagations, rules, and rule-based transformations. To troubleshoot these components, see the troubleshooting documentation in *Oracle Streams Concepts and Administration*.

This section describes common problems and solutions for XStream.

This section contains the following topics:

- An OCI Client Application Cannot Attach to the Outbound Server
- Changes Are Failing to Reach the Client Application in XStream Out
- LCRs Streaming from an Outbound Server Are Missing Extra Attributes
- The XStream Out Client Application Is Unresponsive
- XStream In Cannot Identify an Inbound Server
- Changes Are Not Being Applied by an Inbound Server

An OCI Client Application Cannot Attach to the Outbound Server

An XStream client application cannot attach to an outbound server using the Oracle Call Interface (OCI) `OCIXStreamOutAttach()` function.

The following sections describe possible problems and their solutions.

Problem 1: Client Application Not Connected as Connect User

The client application is not connected as the outbound server's connect user to the outbound server's database. The client application connected to the database as a different user.

Solution 1

To correct problem 1:

- Modify the client application to connect to the database as the connect user before attaching to the outbound server.

Problem 2: Client Application Not Passing Service Handle

The client application is not passing a service handle to the outbound server.

Solution 2

To correct problem 2:

- Modify the client application so that it passes a service handle using `OCISvcCtx` and not `OCIServer`.

See Also:

- "XStream and Security" on page 2-18
- "OCIXStreamOutAttach()" on page 11-60

Changes Are Failing to Reach the Client Application in XStream Out

In an XStream Out configuration, database changes that should be captured and streamed to the XStream client application are not reaching the client application.

The following sections describe possible problems and their solutions.

Problem 1: Capture Process Has Fallen Behind

The capture process has fallen behind.

To determine whether the capture process has fallen behind:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A15
COLUMN CREATE_MESSAGE HEADING 'Last LCR|Create Time'
COLUMN ENQUEUE_MESSAGE HEADING 'Last|Enqueue Time'

SELECT CAPTURE_NAME,
       TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_
MESSAGE,
       TO_CHAR(ENQUEUE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') ENQUEUE_
MESSAGE
  FROM V$STREAMS_CAPTURE;
```

   This query displays the time when the capture process last created a logical change record (LCR) and the time when the capture process last enqueued an LCR. If the times returned are before the time when the database changes were made, then the capture process must catch up and capture the changes.

Solution 1

No action is required. Normally, the capture process will catch up on its own without the need for intervention.

See Also:

- "V$XSTREAM_CAPTURE" on page 13-6
- *Oracle Streams Replication Administrator's Guide*

Problem 2: Rules or Rule-Based Transformation Excluding Changes

Rules or rule-based transformations are excluding the changes that should be captured.

Rules determine which LCRs are captured by a capture process, sent from a source queue to a destination queue by a propagation, and sent to an XStream client application by an outbound server. If the rules are not configured properly, then the client application might not receive the LCRs it should receive. The client application might also receive LCRs that it should not receive.

Rule-based transformations modify the contents of LCRs. Therefore, if the expected change data is not reaching the client application, it might be because a rule-based transformation modified the data or deleted the data. For example, a DELETE_COLUMN declarative rule-based transformation removes a column from an LCR.

Solution 2

To correct problem 2:

- Check the rules and rule-based transformations that are configured for each component in the stream from the capture process to the client application, and correct any problems.

See Also: *Oracle Streams Concepts and Administration*

Problem 3: LCRs Blocked in the Stream

If the capture process has not fallen behind, and there are no problems with rules or rule-based transformations, then LCRs might be blocked in the stream for some other reason. For example, a propagation or outbound server might be disabled, a database link might be broken, or there might be another problem.

You can track an LCR through a stream using one of the following methods:

- Setting the `message_tracking_frequency` capture process parameter to `1` or another relatively low value
- Running the `SET_MESSAGE_TRACKING` procedure in the `DBMS_STREAMS_ADM` package

After using one of these methods, use the `V$STREAMS_MESSAGE_TRACKING` view to monitor the progress of LCRs through the stream. By tracking an LCR through the stream, you can determine where the LCR is blocked.

Solution 3

To correct problem 3:

- Take the appropriate action based on the reason that the LCR is blocked. For example, if a propagation is disabled, then enable it.

See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about tracking LCRs through a stream
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `message_tracking_frequency` capture process parameter
- *Oracle Streams Concepts and Administration* about troubleshooting Oracle Streams components

LCRs Streaming from an Outbound Server Are Missing Extra Attributes

LCRs streaming from an outbound server are expected to include extra attributes, but these attributes are not included in the LCRs.

LCRs can contain the following extra attributes related to database changes:

- `row_id`
- `serial#`
- `session#`
- `thread#`
- `tx_name`
- `username`

By default, a capture process does not capture these extra attributes. If you want extra attributes to be included in LCRs streamed from an outbound server to an XStream client application, but the LCRs do not contain values for extra attributes, then make sure the capture process that captures changes for the outbound server is configured to capture values for the extra attributes.

The following sections describe the possible problem and its solution.

Problem: Capture Process Not Configured to Capture Extra Attributes

The capture process is not configured to capture the required extra attributes.

To display the extra attributes currently being captured by the capture processes in a database:

1. Connect to the database running the capture process as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

   ```
   COLUMN CAPTURE_NAME HEADING 'Capture Process' FORMAT A30
   COLUMN ATTRIBUTE_NAME HEADING 'Attribute Name' FORMAT A30

   SELECT CAPTURE_NAME, ATTRIBUTE_NAME
     FROM DBA_CAPTURE_EXTRA_ATTRIBUTES
     WHERE INCLUDE = 'YES'
     ORDER BY CAPTURE_NAME;
   ```

   If an extra attribute is not displayed by this query, then it is not being captured.

Solution

To solve the problem, configure the capture process to capture the required extra attributes:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package.

Example 7–1 Including the tx_name Attribute for the Capture Process xcapture

```
BEGIN
  DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE(
    capture_name   => 'xcapture',
    attribute_name => 'tx_name',
    include        => TRUE);
END;
/
```

See Also: *Oracle Streams Concepts and Administration*

The XStream Out Client Application Is Unresponsive

The XStream client application in an XStream Out configuration is unresponsive.

The following sections describe the possible problem and its solution.

Problem 1: Streams Pool Size Is Too Small

The Streams pool size might be too small.

To determine whether the Streams pool size is too small:

1. Connect to the outbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following queries at the database that contains the outbound server:
   - Query the `V$PROPAGATION_RECEIVER` view.:

     ```
     SELECT STATE FROM V$PROPAGATION_RECEIVER;
     ```

     If the state is `WAITING FOR MEMORY`, then consider increasing the Streams pool size.
   - Query the `V$STREAMS_POOL_STATISTICS` view.:

     ```
     SELECT TOTAL_MEMORY_ALLOCATED/CURRENT_SIZE FROM V$STREAMS_POOL_STATISTICS;
     ```

     If the value returned is .90 or greater, then consider increasing the Streams pool size.
   - If the outbound server receives changes from a capture process that is running on the same database, then query the `V$STREAMS_CAPTURE` view.:

     ```
     SELECT STATE FROM V$STREAMS_CAPTURE;
     ```

     If the state is `WAITING FOR BUFFER QUEUE TO SHRINK`, then increase the Streams pool size.

Solution 1

To correct problem 1:

- Increase the Streams pool size by modifying the `STREAMS_POOL_SIZE` initialization parameter or by modifying other initialization parameters related to memory.

  See Also:

  - "V$XSTREAM_CAPTURE" on page 13-6
  - *Oracle Streams Replication Administrator's Guide*
  - *Oracle Database Administrator's Guide* for information about setting initialization parameters

Problem 2: Programming Errors

If there is enough memory in the Streams pool, then check your client application for programming errors.

Solution 2

To correct problem 2:

- Correct the programming errors.

XStream In Cannot Identify an Inbound Server

If an XStream In configuration cannot identify an inbound server, then the following error is returned:

```
ORA-26840: STREAMS unable to identify an apply for the source database "%s"
```

The following sections describe the possible problem and its solution.

Problem: Multiple Subscribers to the Inbound Server's Queue

The `ORA-26840` error indicates that there are multiple subscribers to the queue used by the inbound server. Subscribers can include inbound servers, outbound servers, apply processes, and propagations.

To determine whether there are multiple subscribers to the inbound server's queue:

1. Connect to the inbound server database as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Run the following query:

   ```
   SELECT APPLY_NAME SUBSCRIBER, QUEUE_NAME FROM DBA_APPLY UNION
     SELECT PROPAGATION_NAME, SOURCE_QUEUE_NAME
     FROM DBA_PROPAGATION
     ORDER BY QUEUE_NAME;
   ```

   You can add a `WHERE` clause to the query to limit the output to the inbound server's queue.

Solution

To correct the problem:

- If the query returns multiple subscribers to the inbound server's queue, then reconfigure the subscribers so that the inbound server is the only subscriber.

  See Also: Chapter 4, "Configuring XStream"

Changes Are Not Being Applied by an Inbound Server

In an XStream In configuration, database changes are sent in the form of LCRs from the XStream client application to an inbound server.

The following sections describe the possible problem and its solution.

Problem: LCRs Blocked During Apply

If the inbound server is not applying the changes, then the LCRs are blocked during apply. For example, the inbound server might be disabled, an apply handler might be processing LCRs incorrectly, or there might be another problem.

You can track an LCR during apply by an inbound server using the following methods:

- If the client application uses the XStream OCI API, then use the `OCI_LCR_ATTR_MESSAGE_TRACKING_LABEL` attribute in the `OCILCRAttributesSet` function to set tracking label, and use the `OCIXStreamInLCRSend` function to send LCRs.

  The following is sample code for setting message tracking for LCRs using the OCI API:

  ```
  static void set_msg_tracking(myctx_t  *ctx, void *lcr, oratext *label)
  {
   oci_t      *ocip = ctx->ocip;
   oratext    *attr_names[] = {OCI_LCR_ATTR_MESSAGE_TRACKING_LABEL};
   ub2         attr_names_lens[] =

  {sizeof(OCI_LCR_ATTR_MESSAGE_TRACKING_LABEL)-1};
   ub2         dty[] = {SQLT_CHR};
   OCIInd      ind[] = {OCI_IND_NOTNULL};
   void       *data[1];
   ub2         data_lens[1];

   data[0] = label;
   data_lens[0] = strlen(label);
  ```

```
 OCICALL(ocip,
         OCILCRAttributesSet(ocip->svcp, ocip->errp, 1, attr_names,
                             attr_names_lens, dty, data, ind, data_lens,
                             lcr, OCI_DEFAULT));
}

/*----------------------------------------------------------------------
* send_lcr - Send the given lcr and bump up lcr position.
*---------------------------------------------------------------------*/
static void send_lcr(myctx_t  *ctx, void *lcr, ub1 lcrtype)
{
 oci_t      *ocip = ctx->ocip;

 set_msg_tracking(ctx, lcr, "tracking_label");

 OCICALL(ocip,
         OCIXStreamInLCRSend(ocip->svcp, ocip->errp, lcr, lcrtype,
                             0, OCI_DEFAULT));
 ctx->lcr_pos++;
}
```

Using this sample code, to enable message tracking, add the `set_msg_tracking` procedure to the client application, call `set_msg_tracking` from the `send_lcr` procedure when LCRs should be tracked. Replace *tracking_label* with the string you want to use for the tracking label.

- If the client application uses the XStream Java API, then use the `setMessageTrackingLabel` method.

  The following is sample code for setting message tracking for LCRs using the Java API:

  ```
  {
     ((AbstractLCR)lcr).setMessageTrackingLabel("tracking_label");
  }
  ```

  Replace *tracking_label* with the string you want to use for the tracking label.

After using one of these methods, use the `V$STREAMS_MESSAGE_TRACKING` view to monitor the progress of LCRs through a stream. By tracking an LCR, you can determine where the LCR is blocked.

Note: If the LCRs originated from an XStream Out configuration, then the easiest way to track the LCRs is by using the methods described in "Changes Are Failing to Reach the Client Application in XStream Out" on page 7-2.

Solution

To correct the problem:

- Take the appropriate action based on the reason that the LCR is blocked. For example, if the inbound server is disabled, then enable it.

How to Get More Help with XStream

You can check My Oracle Support at `http://support.oracle.com` for more solutions to your problem.

You can visit `http://www.oracle.com/support/contact.html` for more information about Oracle Support.

Part III

XStream PL/SQL Packages Reference

This part contains the XStream PL/SQL packages reference. This part contains the following chapters:

8 DBMS_XSTREAM_ADM

This `DBMS_XSTREAM_ADM` package provides interfaces for streaming database changes between an Oracle database and other systems. XStream enables applications to stream out or stream in database changes.

This chapter contains the following topic:

See Also:

- Chapter 2, "XStream Concepts"
- Part IV, "XStream OCI API Reference"
- *Oracle Database XStream Java API Reference*
- *Oracle Database PL/SQL Packages and Types Reference*

Using DBMS_XSTREAM_ADM

This section contains topics which relate to using the `DBMS_XSTREAM_ADM` package.

- Overview
- Security Model
- Operational Notes

Overview

The package provides interfaces for configuring outbound servers that stream database changes from an Oracle database to other systems. The package also provides interfaces for configuring inbound servers that stream database changes from other systems to an Oracle database. In both cases, the database changes are encapsulated in logical change records (LCRs). Also, the other systems can be Oracle systems or a non-Oracle systems, such as non-Oracle databases or file systems.

XStream outbound servers can stream out LCRs from an Oracle database programmatically using C or Java. After receiving the LCRs, the other system can process them in any customized way. For example, the other system can save the contents of the LCRs to a file, send the LCRs to an Oracle database through an XStream inbound server, or generate SQL statements and execute them on any Oracle or non-Oracle databases.

XStream inbound servers accept LCRs from another system and either apply them to an Oracle database or process them in a customized way using apply handlers.

See Also: Chapter 2, "XStream Concepts"

Security Model

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an XStream administrator and connect as the XStream administrator when using this package.

An administrator must be granted the `DBA` role when the administrator is performing any of the following actions:

- Running the `ADD_OUTBOUND` procedure while connected as a user that is different from the configured connect user for an outbound server
- Running the `ALTER_OUTBOUND` procedure to change the capture user for a capture process or the connect user for an outbound server
- Running the `CREATE_OUTBOUND` procedure, because this procedure creates a capture process
- Running the `ALTER_INBOUND` procedure to change the apply user for an inbound server
- Running the `ADD_INBOUND` procedure while connected as a user that is different from the configured apply user for an inbound server

When the administrator does not need to perform the preceding tasks, the `DBA` role is not required.

See Also:

- "Granting Privileges for the XStream Administrator" on page 4-1
- "XStream and Security" on page 2-18 for more information about XStream and security

Operational Notes

Some subprograms in the `DBMS_APPLY_ADM` package can manage XStream outbound servers, and some subprograms in the `DBMS_APPLY_ADM` package can manage XStream inbound servers.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for details about which subprograms can manage outbound servers and inbound servers

Summary of DBMS_XSTREAM_ADM Subprograms

Table 8–1 DBMS_XSTREAM_ADM Package Subprograms

Subprogram	Description
ADD_OUTBOUND Procedure on page 8-7	Creates an XStream outbound server that dequeues LCRs from the specified queue
ADD_SUBSET_OUTBOUND_RULES Procedure on page 8-11	Adds subset rules to an outbound server configuration
ALTER_INBOUND Procedure on page 8-13	Modifies an XStream inbound server
ALTER_OUTBOUND Procedure on page 8-14	Modifies an XStream outbound server
CREATE_INBOUND Procedure on page 8-19	Creates an XStream inbound server and its queue
CREATE_OUTBOUND Procedure on page 8-21	Creates an XStream outbound server, queue, and capture process to enable XStream client applications to stream out Oracle database changes encapsulated in LCRs
DROP_INBOUND Procedure on page 8-25	Removes an inbound server configuration
DROP_OUTBOUND Procedure on page 8-26	Removes an outbound server configuration
ENABLE_GG_XSTREAM_FOR_STREAMS Procedure on page 8-27	Enables XStream performance optimizations for Oracle Streams components
IS_GG_XSTREAM_FOR_STREAMS Function on page 8-29	Returns `TRUE` if XStream performance optimizations are enabled for Oracle Streams components, or returns `FALSE` if XStream performance optimizations are disabled for Oracle Streams components
REMOVE_SUBSET_OUTBOUND_RULES Procedure on page 8-30	Removes subset rules from an outbound server configuration

Note: All subprograms commit unless specified otherwise.

ADD_OUTBOUND Procedure

This procedure creates an XStream outbound server that dequeues LCRs from the specified queue. The outbound server streams out the LCRs to an XStream client application.

This procedure creates neither a capture process nor a queue. To create an outbound server, a capture process, and a queue with one procedure call, use the CREATE_OUTBOUND Procedure.

To create the capture process individually, use one of the following packages:

- `DBMS_STREAMS_ADM`
- `DBMS_CAPTURE_ADM`

To create a queue individually, use the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package.

This procedure is overloaded. One `table_names` parameter is type `VARCHAR2` and the other `table_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. Also, one `schema_names` parameter is type `VARCHAR2` and the other `schema_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the lists of tables and schemas in different ways and are mutually exclusive.

Note:

- A client application can create multiple sessions. Each session can attach to only one outbound server, and each outbound server can serve only one session at a time. However, different client application sessions can connect to different outbound servers. See Part IV, "XStream OCI API Reference" and *Oracle Database XStream Java API Reference* for information about attaching to an outbound server.
- This procedure enables the outbound server that it creates.
- Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `capture_name`, `start_scn`, and `start_time` parameters are included in this procedure.

Syntax

```
DBMS_XSTREAM_ADM.ADD_OUTBOUND(
   server_name     IN  VARCHAR2,
   queue_name      IN  VARCHAR2   DEFAULT NULL,
   source_database IN  VARCHAR2   DEFAULT NULL,
   table_names     IN  DBMS_UTILITY.UNCL_ARRAY,
   schema_names    IN  DBMS_UTILITY.UNCL_ARRAY,
   connect_user    IN  VARCHAR2   DEFAULT NULL,
   comment         IN  VARCHAR2   DEFAULT NULL,
   capture_name    IN  VARCHAR2   DEFAULT NULL,
   start_scn       IN  NUMBER     DEFAULT NULL,
   start_time      IN  TIMESTAMP  DEFAULT NULL);

DBMS_XSTREAM_ADM.ADD_OUTBOUND(
   server_name     IN  VARCHAR2,
   queue_name      IN  VARCHAR2   DEFAULT NULL,
   source_database IN  VARCHAR2   DEFAULT NULL,
   table_names     IN  VARCHAR2   DEFAULT NULL,
   schema_names    IN  VARCHAR2   DEFAULT NULL,
   connect_user    IN  VARCHAR2   DEFAULT NULL,
   comment         IN  VARCHAR2   DEFAULT NULL,
   capture_name    IN  VARCHAR2   DEFAULT NULL,
   start_scn       IN  NUMBER     DEFAULT NULL,
   start_time      IN  TIMESTAMP  DEFAULT NULL);
```

Parameters

Table 8–2 ADD_OUTBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the outbound server being created. A `NULL` specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing outbound server, inbound server, apply process, or messaging client. **Note:** The `server_name` setting cannot be altered after the outbound server is created.
`queue_name`	The name of the local queue from which the outbound server dequeues LCRs, specified as `[`*`schema_name`*`.]`*`queue_name`*. The current database must contain the queue, and the queue must be `ANYDATA` type. For example, to specify a queue named `xstream_queue` in the `xstrmadmin` schema, enter `xstrmadmin.xstream_queue` for this parameter. If the schema is not specified, then the current user is the default. If `NULL`, the procedure raises an error.
`source_database`	The global name of the source database. The source database is where the changes being captured originated. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify `DBS1` and the domain is `EXAMPLE.COM`, then the procedure specifies `DBS1.EXAMPLE.COM` automatically. If `NULL`, then this procedure does not add a condition regarding the source database to the generated rules. Otherwise, a condition regarding the source database is added.
`table_names`	The tables for which data manipulation language (DML) and data definition language (DDL) changes are streamed out to the XStream client application. The tables can be specified in the following ways: ■ Comma-delimited list of type `VARCHAR2`. ■ A PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a table. Specify the first table in position 1. The last position must be `NULL`. Each table should be specified as `[`*`schema_name`*`.]`*`table_name`*. For example, you can specify `hr.employees`. If the schema is not specified, then the current user is the default. **See Also:** "Usage Notes" on page 8-9 for more information about this parameter

Table 8–2 (Cont.) ADD_OUTBOUND Procedure Parameters

Parameter	Description
`schema_names`	The schemas for which DML and DDL changes are streamed out to the XStream client application. The schemas can be specified in the following ways: ■ Comma-delimited list of type `VARCHAR2`. ■ A PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a schema. Specify the first schema in position 1. The last position must be `NULL`. **Note:** This procedure does not concatenate the `schema_names` parameter with the `table_names` parameter. To specify tables, enter fully qualified table names in the `table_names` parameter (*`schema_name.table_name`*). **See Also:** "Usage Notes" on page 8-9 for more information about this parameter
`connect_user`	The user who can attach to the specified outbound server to retrieve the LCR stream. The client application must attach to the outbound server as the specified connect user. See "CREATE_OUTBOUND Procedure" on page 8-21 for information about the privileges required by a connect user. If `NULL`, then the current user is the default.
`comment`	An optional comment associated with the outbound server.
`capture_name`	The name of the capture process configured to capture changes for the outbound server. Do not specify an owner. If the specified name matches the name of an existing capture process for another outbound server, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set. If the specified name matches the name of an existing capture process for an apply process, then an error is raised. If the specified name does not match the name of an existing capture process, then an error is raised. If `NULL`, then the outbound server is created without a capture process.
`start_scn`	A valid system change number (SCN) for the database from which the capture process starts capturing changes. If the `capture_name` parameter is `NULL`, then this parameter is ignored. If `NULL` and the `capture_name` parameter is non-`NULL`, then the start SCN of the capture process is not changed. An error is returned if an invalid SCN is specified. The `start_scn` and `start_time` parameters are mutually exclusive.
`start_time`	A valid time from which the capture process starts capturing changes. If the `capture_name` parameter is `NULL`, then this parameter is ignored. If `NULL` and the `capture_name` parameter is non-`NULL`, then the start SCN of the capture process is not changed. The `start_scn` and `start_time` parameters are mutually exclusive.

Usage Notes

The following list describes the behavior of the outbound server for various combinations of the `table_names` and `schema_names` parameters:

- If both the `table_names` and `schema_names` parameters are `NULL` or empty, then the outbound server streams all DML and DDL changes to the client application.

This procedure is overloaded. The `table_names` and `schema_names` parameters are defaulted to `NULL`. Do not specify `NULL` for both `table_names` and `schema_names` in the same call; otherwise, error `PLS-00307` is returned.

- If both the `table_names` and `schema_names` parameters are specified, then the outbound server streams DML and DDL changes for the specified tables and schemas.
- If the `table_names` parameter is specified and the `schema_names` parameter is `NULL` or empty, then the outbound server streams DML and DDL changes for the specified tables.
- If the `table_names` parameter is `NULL` or empty and the `schema_names` parameter is specified, then the outbound server streams DML and DDL changes for the specified schemas.

For the procedure that uses the `DBMS_UTILITY.UNCL_ARRAY` type for the `table_names` and `schema_names` parameters, both parameters must be specified. To specify only tables, the `schema_names` parameter must be specified and empty. To specify only schemas, the `table_names` parameter must be specified and empty.

Note: An empty array includes one `NULL` entry.

ADD_SUBSET_OUTBOUND_RULES Procedure

This procedure adds subset rules to an outbound server configuration. Subset rules instruct the outbound server to stream out a subset of the changes to the specified tables. Outbound servers can stream out a subset of both rows and columns.

This procedure is overloaded. One `column_list` parameter is type `VARCHAR2` and the other `column_list` parameter is type `DBMS_UTILITY.LNAME_ARRAY`. These parameters enable you to enter the list of columns in different ways and are mutually exclusive.

Note: This procedure does not add rules to the outbound server's capture process.

Syntax

```
DBMS_XSTREAM_ADM.ADD_SUBSET_OUTBOUND_RULES(
   server_name IN VARCHAR2,
   table_name  IN VARCHAR2,
   condition   IN VARCHAR2  DEFAULT NULL,
   column_list IN DBMS_UTILITY.LNAME_ARRAY,
   keep        IN BOOLEAN   DEFAULT TRUE);

DBMS_XSTREAM_ADM.ADD_SUBSET_OUTBOUND_RULES(
   server_name IN VARCHAR2,
   table_name  IN VARCHAR2,
   condition   IN VARCHAR2  DEFAULT NULL,
   column_list IN VARCHAR2  DEFAULT NULL,
   keep        IN BOOLEAN   DEFAULT TRUE);
```

Parameters

Table 8–3 ADD_SUBSET_OUTBOUND_RULES Procedure Parameters

Parameter	Description
`server_name`	The name of the outbound server to which rules are being added. Specify an existing outbound server. Do not specify an owner.
`table_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, you can specify `hr.employees`. If the schema is not specified, then the current user is the default. If the outbound server configuration uses a local capture process, then the table must exist at the local source database. If the outbound server configuration uses a downstream capture process, then the table must exist at both the source database and at the downstream capture database. The specified table cannot have any LOB, `LONG`, or `LONG RAW` columns currently or in the future.
`condition`	The subset condition. Specify this condition similar to the way you specify conditions in a `WHERE` clause in SQL. For example, to specify rows in the `hr.employees` table where the `salary` is greater than `4000` and the `job_id` is `SA_MAN`, enter the following as the condition: `' salary > 4000 and job_id = ''SA_MAN'' '` If `NULL`, then the procedure raises an error. **Note:** The quotation marks in the preceding example are all single quotation marks.

Table 8–3 (Cont.) ADD_SUBSET_OUTBOUND_RULES Procedure Parameters

Parameter	Description
`column_list`	The list of columns either to include in the outbound server configuration or to exclude from the outbound server configuration. Whether the columns are included or excluded depends on the setting for the `keep` parameter. The columns can be specified in the following ways: ■ Comma-delimited list of type `VARCHAR2`. ■ A PL/SQL associative array of type `DBMS_UTILITY.LNAME_ARRAY`, where each element is the name of a column. Specify the first column in position 1. The last position must be `NULL`. To include or exclude all of the columns in a table, specify each column in the table in the list or array. If `NULL`, then the procedure raises an error.
`keep`	If `TRUE`, then the columns specified in the `column_list` parameter are kept as part of the outbound server configuration. Therefore, changes to these columns that satisfy the condition in the `condition` parameter are streamed to the outbound server's client application. If `FALSE`, then the columns specified in the `column_list` parameter are excluded from the outbound server configuration. Therefore, changes to these columns are not streamed to the outbound server's client application. **See Also:** "Usage Notes" on page 8-12

Usage Notes

When the `keep` parameter is set to `TRUE`, this procedure creates a keep columns declarative rule-based transformation for the columns listed in `column_list`.

When the `keep` parameter is set to `FALSE`, this procedure creates a delete column declarative rule-based transformation for each column listed in `column_list`.

See Also: *Oracle Streams Concepts and Administration* for information about declarative rule-based transformations

ALTER_INBOUND Procedure

This procedure modifies an XStream inbound server.

Syntax

```
DBMS_XSTREAM_ADM.ALTER_INBOUND(
   server_name IN VARCHAR2,
   apply_user  IN VARCHAR2  DEFAULT NULL,
   comment     IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 8–4 ALTER_INBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the inbound server being altered. Specify an existing inbound server. Do not specify an owner.
`apply_user`	The user who applies all DML and DDL changes that satisfy the inbound server rule sets, who runs user-defined apply handlers, and who runs custom rule-based transformations configured for inbound server rules. The client application must attach to the inbound server as the apply user. Specify a user to change the apply user. In this case, the user who invokes the `ALTER_INBOUND` procedure must be granted the `DBA` role. Only the `SYS` user can set the `apply_user` to `SYS`. If `NULL`, then the apply user is not changed. See "CREATE_INBOUND Procedure" on page 8-19 for information about the required privileges for an apply user.
`comment`	An optional comment associated with the inbound server. If non-`NULL`, then the specified comment replaces the existing comment. If `NULL`, then the existing comment is not changed.

ALTER_OUTBOUND Procedure

This procedure modifies an XStream outbound server configuration.

This procedure always alters the specified outbound server. This procedure can also alter the outbound server's capture process when either of the following conditions is met:

- The capture process was created by the `CREATE_OUTBOUND` procedure in this package.
- The queue used by the capture process was created by the `CREATE_OUTBOUND` procedure.

To check whether this procedure can alter the outbound server's capture process, query the `CAPTURE_NAME` column in the `DBA_XSTREAM_OUTBOUND` view. When the name of the capture process appears in the `CAPTURE_NAME` column of this view, the `ALTER_OUTBOUND` procedure can manage the capture process's rules or change the capture user for the capture process. When the `CAPTURE_NAME` column of this view is `NULL`, the `ALTER_OUTBOUND` procedure cannot manage the capture process.

This procedure is overloaded. One `table_names` parameter is type `VARCHAR2` and the other `table_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. Also, one `schema_names` parameter is type `VARCHAR2` and the other `schema_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of tables and schemas in different ways and are mutually exclusive.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `start_scn` and `start_time` parameters are included in this procedure.

Syntax

```
DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
   server_name    IN VARCHAR2,
   table_names    IN DBMS_UTILITY.UNCL_ARRAY,
   schema_names   IN DBMS_UTILITY.UNCL_ARRAY,
   add            IN BOOLEAN    DEFAULT TRUE,
   capture_user   IN VARCHAR2   DEFAULT NULL,
   connect_user   IN VARCHR2    DEFAULT NULL,
   comment        IN VARCHAR2   DEFAULT NULL,
   inclusion_rule IN BOOLEAN    DEFAULT TRUE,
   start_scn      IN NUMBER     DEFAULT NULL,
   start_time     IN TIMESTAMP  DEFAULT NULL);

DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
   server_name    IN VARCHAR2,
   table_names    IN VARCHAR2   DEFAULT NULL,
   schema_names   IN VARCHAR2   DEFAULT NULL,
   add            IN BOOLEAN    DEFAULT TRUE,
   capture_user   IN VARCHAR2   DEFAULT NULL,
   connect_user   IN VARCHAR2   DEFAULT NULL,
   comment        IN VARCHAR2   DEFAULT NULL,
   inclusion_rule IN BOOLEAN    DEFAULT TRUE,
   start_scn      IN NUMBER     DEFAULT NULL,
   start_time     IN TIMESTAMP  DEFAULT NULL);
```

Parameters

Table 8–5 ALTER_OUTBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the outbound server being altered. Specify an existing outbound server. Do not specify an owner.
`table_names`	The tables that are either added to or removed from the XStream Out configuration. Whether the tables are added or removed depends on the setting for the `add` parameter. The tables can be specified in the following ways: ▪ Comma-delimited list of type `VARCHAR2`. ▪ A PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a table. Specify the first table in position 1. The last position must be `NULL`. Each table should be specified as `[`*`schema_name`*`.]`*`table_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. **See Also:** "Usage Notes" on page 8-18 for more information about this parameter
`schema_names`	The schemas that are either added to or removed from the XStream Out configuration. Whether the schemas are added or removed depends on the setting for the `add` parameter. The schemas can be specified in the following ways: ▪ Comma-delimited list of type `VARCHAR2`. ▪ A PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a schema. Specify the first schema in position 1. The last position must be `NULL`. **Note:** This procedure does not concatenate the `schema_names` parameter with the `table_names` parameter. To specify tables, enter fully qualified table names in the `table_names` parameter (*`schema_name`*`.`*`table_name`*). **See Also:** "Usage Notes" on page 8-18 for more information about this parameter
`add`	If `TRUE`, then the procedure adds to the XStream Out configuration the tables specified in the `table_names` parameter and the schemas specified in the `schema_names` parameter. If `FALSE`, then the procedure removes from the XStream Out configuration the tables specified in the `table_names` parameter and the schemas specified in the `schema_names` parameter.

Table 8–5 (Cont.) ALTER_OUTBOUND Procedure Parameters

Parameter	Description
`capture_user`	The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. Specify a user to change the capture user. In this case, the user who invokes the `ALTER_OUTBOUND` procedure must be granted the `DBA` role. Only the `SYS` user can set the `capture_user` to `SYS`. If `NULL`, then the capture user is not changed. If you change the capture user, then this procedure grants the new capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user. Ensure that the capture user is granted the other required privileges. See "CREATE_OUTBOUND Procedure" on page 8-21 for information about the privileges required by a capture user. The capture process is stopped and restarted automatically when you change the value of this parameter. **Note:** If the capture user for a capture process is dropped using `DROP USER . . . CASCADE`, then the capture process is also dropped automatically.
`connect_user`	The user who can attach to the specified outbound server to retrieve the change stream. The XStream client application must attach to the outbound server as the specified connect user. Specify a user to change the connect user. In this case, the user who invokes the `ALTER_OUTBOUND` procedure must be granted the `DBA` role. Only the `SYS` user can set the `connect_user` to `SYS`. If `NULL`, then the connect user is not changed. If you change the connect user, then this procedure grants the new connect user dequeue privileges on the queue used by the outbound server and configures the user as a secure queue user. Ensure that the connect user is granted the other required privileges. See "CREATE_OUTBOUND Procedure" on page 8-21 for information about the privileges required by a connect user.
`comment`	An optional comment associated with the outbound server. If non-`NULL`, then the specified comment replaces the existing comment. If `NULL`, then the existing comment is not changed.

Table 8–5 (Cont.) ALTER_OUTBOUND Procedure Parameters

Parameter	Description
`inclusion_rule`	If `TRUE` and the `add` parameter is set to `TRUE`, then the procedure adds rules for the tables specified in the `table_names` parameter and the schemas specified in the `schema_names` parameter to the positive rule sets in the XStream Out configuration. When rules for tables and schemas are in positive rule sets, the XStream Out configuration streams DML and DDL changes to the tables and schemas out to the client application. If `TRUE` and the `add` parameter is set to `FALSE`, then the procedure removes rules for the tables specified in the `table_names` parameter and the schemas specified in the `schema_names` parameter from the positive rule sets in the XStream Out configuration. If `FALSE` and the `add` parameter is set to `TRUE`, then the procedure adds rules for the tables specified in the `table_names` parameter and the schemas specified in the `schema_names` parameter to the negative rule sets in the XStream Out configuration. When rules for tables and schemas are in negative rule sets, the XStream Out configuration does not stream changes to the tables and schemas out to the client application. If `FALSE` and the `add` parameter is set to `FALSE`, then the procedure removes rules for the tables specified in the `table_names` parameter and the schemas specified in the `schema_names` parameter from the negative rule sets in the XStream Out configuration.
`start_scn`	A valid SCN for the database from which the capture process starts capturing changes. To be valid, the SCN value must be greater than or equal to the first SCN for the capture process. If a valid SCN is specified, then the capture process captures changes from the specified SCN when it is restarted. An error is returned if an invalid SCN is specified. If `NULL` and the `start_time` parameter is `NULL`, then the start SCN is not changed. If `NULL` and the `start_time` parameter is non-`NULL`, then the start SCN is changed to match the specified start time. The `start_scn` and `start_time` parameters are mutually exclusive. **Note:** If the capture process is enabled, then the `ALTER_OUTBOUND` procedure automatically stops and restarts the capture process when the `start_scn` parameter is non-`NULL`. If the capture process is disabled, then the `ALTER_OUTBOUND` procedure automatically starts the capture process when the `start_scn` parameter is non-`NULL`.
`start_time`	A valid time from which the capture process starts capturing changes. To be valid, the time must correspond to an SCN value that is greater than or equal to the first SCN for the capture process. If a valid time is specified, then the capture process captures changes from the specified time when it is restarted. An error is returned if an invalid time is specified. If `NULL` and the `start_scn` parameter is `NULL`, then the start time is not changed. If `NULL` and the `start_scn` parameter is non-`NULL`, then the start time is changed to match the specified start SCN. The `start_scn` and `start_time` parameters are mutually exclusive. **Note:** If the capture process is enabled, then the `ALTER_OUTBOUND` procedure automatically stops and restarts the capture process when the `start_time` parameter is non-`NULL`. If the capture process is disabled, then the `ALTER_OUTBOUND` procedure automatically starts the capture process when the `start_time` parameter is non-`NULL`.

Usage Notes

The following list describes the behavior of the outbound server for various combinations of the table_names and schema_names parameters:

- If both the table_names and schema_names parameters are NULL or empty, then no rules are changed for the XStream Out configuration.

  This procedure is overloaded. The table_names and schema_names parameters are defaulted to NULL. Do not specify NULL for both table_names and schema_names in the same call; otherwise, error PLS-00307 is returned.

- If both the table_names and schema_names parameters are specified, then the rules for the tables and schemas are added to or removed from the XStream Out configuration, depending on the setting of the add parameter.
- If the table_names parameter is specified and the schema_names parameter is NULL or empty, then the rules for the tables are added to or removed from the XStream Out configuration, depending on the setting of the add parameter. The existing rules for schemas are not changed for the XStream Out configuration.
- If the table_names parameter is NULL or empty and the schema_names parameter is specified, then the rules for the schemas are added to or removed from the XStream Out configuration, depending on the setting of the add parameter. The existing rules for tables are not changed for the XStream Out configuration.

For the procedure that uses the DBMS_UTILITY.UNCL_ARRAY type for the table_names and schema_names parameters, both parameters must be specified. To specify only tables, the schema_names parameter must be specified and empty. To specify only schemas, the table_names parameter must be specified and empty.

Note: An empty array includes one NULL entry.

CREATE_INBOUND Procedure

This procedure creates an XStream inbound server and its queue.

Note: A client application can create multiple sessions. Each session can attach to only one inbound server, and each inbound server can serve only one session at a time. However, different client application sessions can connect to different inbound servers. See Part IV, "XStream OCI API Reference" and *Oracle Database XStream Java API Reference* for information about attaching to an inbound server.

Syntax

```
DBMS_XSTREAM_ADM.CREATE_INBOUND(
   server_name IN VARCHAR2,
   queue_name  IN VARCHAR2,
   apply_user  IN VARCHAR2  DEFAULT NULL,
   comment     IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 8–6 CREATE_INBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the inbound server being created. A `NULL` specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing outbound server, inbound server, apply process, or messaging client. **Note:** The `server_name` setting cannot be altered after the inbound server is created.
`queue_name`	The name of the local queue used by the inbound server, specified as `[`*`schema_name`*`.]`*`queue_name`*. If the specified queue exists, then it is used. If the specified queue does not exist, then the procedure creates it. For example, to specify a queue named `xstream_queue` in the `xstrmadmin` schema, enter `xstrmadmin.xstream_queue` for this parameter. If the schema is not specified, then the current user is the default. **Note:** An inbound server's queue is used only to store error transactions.

Table 8–6 (Cont.) CREATE_INBOUND Procedure Parameters

Parameter	Description
`apply_user`	The apply user. If `NULL`, then the current user is the default. The client application must attach to the inbound server as the apply user. The apply user is the user in whose security domain an inbound server evaluates whether LCRs satisfy its rule sets, applies DML and DDL changes directly to database objects, runs custom rule-based transformations configured for inbound server rules, and runs apply handlers configured for the inbound server. This user must have the necessary privileges to perform these actions. This procedure grants the apply user dequeue privileges on the queue used by the inbound server and configures the user as a secure queue user. In addition to the privileges granted by this procedure, you must grant the following privileges to the apply user: ■ The necessary privileges to perform DML and DDL changes on the apply objects ■ `EXECUTE` privilege on the rule sets used by the inbound server ■ `EXECUTE` privilege on all rule-based transformation functions used in the rule set ■ `EXECUTE` privilege on all apply handler procedures You can grant these privileges directly to the apply user, or you can grant them through roles. In addition, the apply user must be granted `EXECUTE` privilege on all packages, including Oracle supplied packages, that are invoked in subprograms run by the inbound server. These privileges must be granted directly to the apply user. They cannot be granted through roles. **Note:** If the apply user for an inbound server is dropped using `DROP USER . . . CASCADE`, then the inbound server is also dropped automatically.
`comment`	An optional comment associated with the inbound server.

Usage Notes

By default, an inbound server does not use rules or rule sets. Therefore, an inbound server applies all of the LCRs sent to it by an XStream client application. However, to filter the LCRs sent to an inbound server, you can add rules and rule sets to an inbound server using the `DBMS_STREAMS_ADM` and `DBMS_RULE_ADM` packages.

See Also: *Oracle Streams Concepts and Administration*

CREATE_OUTBOUND Procedure

This procedure creates an XStream outbound server, queue, and capture process to enable client applications to stream out Oracle database changes.

This procedure is overloaded. One `table_names` parameter is type `VARCHAR2` and the other `table_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. Also, one `schema_names` parameter is type `VARCHAR2` and the other `schema_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of tables and schemas in different ways and are mutually exclusive.

Note:

- A client application can create multiple sessions. Each session can attach to only one outbound server, and each outbound server can serve only one session at a time. However, different client application sessions can connect to different outbound servers. See "OCIXStreamOutAttach()" on page 11-60 and *Oracle Database XStream Java API Reference* for information about attaching to an outbound server.
- If the `capture_name` parameter is `NULL`, then this procedure automatically generates a name for the capture process that it creates.
- This procedure automatically generates a name for the queue that it creates.
- This procedure enables both the capture process and outbound server that it creates.
- Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `capture_name` parameter is included in this procedure.

Syntax

```
DBMS_XSTREAM_ADM.CREATE_OUTBOUND(
   server_name     IN VARCHAR2,
   source_database IN VARCHAR2  DEFAULT NULL,
   table_names     IN DBMS_UTILITY.UNCL_ARRAY,
   schema_names    IN DBMS_UTILITY.UNCL_ARRAY,
   capture_user    IN VARCHAR2  DEFAULT NULL,
   connect_user    IN VARCHAR2  DEFAULT NULL,
   comment         IN VARCHAR2  DEFAULT NULL,
   capture_name    IN VARCHAR2  DEFAULT NULL);

DBMS_XSTREAM_ADM.CREATE_OUTBOUND(
   server_name     IN VARCHAR2,
   source_database IN VARCHAR2  DEFAULT NULL,
   table_names     IN VARCHAR2  DEFAULT NULL,
   schema_names    IN VARCHAR2  DEFAULT NULL,
   capture_user    IN VARCHAR2  DEFAULT NULL,
   connect_user    IN VARCHAR2  DEFAULT NULL,
   comment         IN VARCHAR2  DEFAULT NULL,
   capture_name    IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 8–7 CREATE_OUTBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the outbound server being created. A `NULL` specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing outbound server, inbound server, apply process, or messaging client. **Note:** The `server_name` setting cannot be altered after the outbound server is created.
`source_database`	The global name of the source database. The source database is where the changes to be captured originated. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify `DBS1` and the domain is `EXAMPLE.COM`, then the procedure specifies `DBS1.EXAMPLE.COM` automatically. If `NULL`, or if the specified name is the same as the global name of the current database, then local capture is assumed. If non-`NULL` and the specified name is different from the global name of the current database, then downstream capture is assumed. In this case, configure the transmission of redo data from the source database to the downstream database before running the `CREATE_OUTBOUND` procedure. See *Oracle Streams Replication Administrator's Guide* for instructions.
`table_names`	The tables for which DML and DDL changes are streamed out to the XStream client application. The tables can be specified in the following ways: ■ Comma-delimited list of type `VARCHAR2`. ■ A PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a table. Specify the first table in position 1. The last position must be `NULL`. Each table should be specified as `[`*`schema_name`*`.]`*`table_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. **See Also:** "Usage Notes" on page 8-24 for more information about this parameter
`schema_names`	The schemas for which DML and DDL changes are streamed out to the XStream client application. The schemas can be specified in the following ways: ■ Comma-delimited list of type `VARCHAR2`. ■ A PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a schema. Specify the first schema in position 1. The last position must be `NULL`. **Note:** This procedure does not concatenate the `schema_names` parameter with the `table_names` parameter. To specify tables, enter fully qualified table names in the `table_names` parameter (*`schema_name.table_name`*). **See Also:** "Usage Notes" on page 8-24 for more information about this parameter

Table 8–7 (Cont.) CREATE_OUTBOUND Procedure Parameters

Parameter	Description
`capture_user`	The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If `NULL`, then the current user is the default. This procedure grants the capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user. In addition, ensure that the capture user has the following privileges: ■ `EXECUTE` privilege on the rule sets used by the capture process ■ `EXECUTE` privilege on all rule-based transformation functions used in the positive rule set You can grant these privileges directly to the apply user, or you can grant them through roles. In addition, the capture user must be granted `EXECUTE` privilege on all packages, including Oracle supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles. Only a user who is granted the `DBA` role can set a capture user. Only the `SYS` user can set the `capture_user` to `SYS`. A capture user does not require privileges on a database object to capture changes made to it. The capture process can pass these changes to a custom rule-based transformation function. Therefore, ensure that you consider security implications when you configure a capture process.
`connect_user`	The user who can attach to the specified outbound server to retrieve the change stream. The client application must attach to the outbound server as the specified connect user. If `NULL`, then the current user is the default. The connect user is the user in whose security domain an outbound server dequeues LCRs that satisfy its rule sets and runs custom rule-based transformations configured for outbound server rules. This user must have the necessary privileges to perform these actions. This procedure grants the connect user dequeue privileges on the queue used by the outbound server and configures the user as a secure queue user. In addition to the privileges granted by this procedure, grant the following privileges to the connect user: ■ `EXECUTE` privilege on the rule sets used by the outbound server ■ `EXECUTE` privilege on all rule-based transformation functions used in the rule set You can grant these privileges directly to the connect user, or you can grant them through roles. In addition, the connect user must be granted `EXECUTE` privilege on all packages, including Oracle supplied packages, that are invoked in subprograms run by the outbound server. These privileges must be granted directly to the apply user. They cannot be granted through roles.
`comment`	An optional comment associated with the outbound server.

Table 8–7 (Cont.) CREATE_OUTBOUND Procedure Parameters

Parameter	Description
`capture_name`	The name of the capture process configured to capture changes for the outbound server. Do not specify an owner. The capture process must not exist. If the specified name matches the name of an existing capture process, then an error is raised. If the name does not match the name of an existing capture process, then the procedure creates a new capture process with the specified name. If `NULL`, then the system creates a new capture process with a system-generated name. **Note:** The capture process name cannot be altered after the capture process is created.

Usage Notes

The following list describes the behavior of the outbound server for various combinations of the `table_names` and `schema_names` parameters:

- If both the `table_names` and `schema_names` parameters are `NULL` or empty, then the outbound server streams all DML and DDL changes to the client application.

  This procedure is overloaded. The `table_names` and `schema_names` parameters are defaulted to `NULL`. Do not specify `NULL` for both `table_names` and `schema_names` in the same call; otherwise, error `PLS-00307` is returned.

- If both the `table_names` and `schema_names` parameters are specified, then the outbound server streams DML and DDL changes for the specified tables and schemas.
- If the `table_names` parameter is specified and the `schema_names` parameter is `NULL` or empty, then the outbound server streams DML and DDL changes for the specified tables.
- If the `table_names` parameter is `NULL` or empty and the `schema_names` parameter is specified, then the outbound server streams DML and DDL changes for the specified schema.

For the procedure that uses the `DBMS_UTILITY.UNCL_ARRAY` type for the `table_names` and `schema_names` parameters, both parameters must be specified. To specify only tables, the `schema_names` parameter must be specified and empty. To specify only schemas, the `table_names` parameter must be specified and empty.

Note: An empty array includes one `NULL` entry.

DROP_INBOUND Procedure

This procedure removes an inbound server configuration.

This procedure always removes the specified inbound server. This procedure also removes the queue for the inbound server if all of the following conditions are met:

- One call to the `CREATE_INBOUND` procedure created the queue.
- The inbound server is the only subscriber to the queue.

See Also: "CREATE_INBOUND Procedure" on page 8-19

Syntax

```
DBMS_XSTREAM_ADM.DROP_INBOUND(
   server_name IN VARCHAR2);
```

Parameters

Table 8–8 DROP_INBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the inbound server being removed. Specify an existing inbound server. Do not specify an owner.

DROP_OUTBOUND Procedure

This procedure removes an outbound server configuration.

This procedure always drops the specified outbound server. This procedure also drops the queue used by the outbound server if both of the following conditions are met:

- The queue was created by the `CREATE_OUTBOUND` procedure in this package.
- The outbound server is the only subscriber to the queue.

If either one of the preceding conditions is not met, then the `DROP_OUTBOUND` procedure only drops the outbound server. It does not drop the queue.

This procedure also drops the capture process for the outbound server if both of the following conditions are met:

- The procedure can drop the outbound server's queue.
- The capture process was created by the `CREATE_OUTBOUND` procedure.

If the procedure can drop the queue but cannot manage the capture process, then it drops the queue without dropping the capture process.

See Also:

- "ADD_OUTBOUND Procedure" on page 8-7
- "CREATE_OUTBOUND Procedure" on page 8-21

Syntax

```
DBMS_XSTREAM_ADM.DROP_OUTBOUND(
   server_name IN VARCHAR2);
```

Parameters

Table 8–9 DROP_OUTBOUND Procedure Parameters

Parameter	Description
`server_name`	The name of the outbound server being removed. Specify an existing outbound server. Do not specify an owner.

ENABLE_GG_XSTREAM_FOR_STREAMS Procedure

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

This procedure enables XStream capabilities and performance optimizations for Oracle Streams components.

This procedure is intended for users of Oracle Streams who want to enable XStream capabilities and optimizations. For example, you can enable the optimizations for an Oracle Streams replication configuration that uses capture processes and apply processes to replicate changes between Oracle databases.

These capabilities and optimizations are enabled automatically for XStream components, such as outbound servers, inbound servers, and capture processes that send changes to outbound servers. It is not necessary to run this procedure for XStream components.

When XStream capabilities are enabled, Oracle Streams components can stream ID key LCRs and sequence LCRs. The XStream performance optimizations improve efficiency in various areas, including:

- LCR processing
- Handling large transactions
- DML execution during apply
- Dependency computation and scheduling
- Capture process parallelism

Syntax

```
DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS(
   enable IN BOOLEAN  TRUE);
```

Parameters

Table 8–10 ENABLE_GG_XSTREAM_FOR_STREAMS Procedure Parameters

Parameter	Description
`enable`	If `TRUE`, then enable XStream performance optimizations for Oracle Streams components. If `FALSE`, then disable XStream performance optimizations for Oracle Streams components.

Usage Notes

The following usage notes apply to this procedure:

- When you run this procedure, all capture processes and apply processes are restarted.
- After you run this procedure, the `PURPOSE` column in the following views displays `XStream Streams`:
  – `ALL_APPLY`

  - `DBA_APPLY`
  - `ALL_CAPTURE`
  - `DBA_CAPTURE`

- A license for the Oracle GoldenGate product is required to enable XStream performance optimizations for Oracle Streams components.

  See Also:

  - "IS_GG_XSTREAM_FOR_STREAMS Function" on page 8-29
  - "Prerequisites for XStream" on page 1-3

IS_GG_XSTREAM_FOR_STREAMS Function

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

This function returns `TRUE` if XStream performance optimizations are enabled for Oracle Streams components, or this function returns `FALSE` if XStream performance optimizations are disabled for Oracle Streams components.

See Also: "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" on page 8-27

Syntax

```
DBMS_XSTREAM_ADM.IS_GG_XSTREAM_FOR_STREAMS
RETURN BOOLEAN;
```

REMOVE_SUBSET_OUTBOUND_RULES Procedure

This procedure removes subset rules from an outbound server configuration.

The names of the specified insert, update, and delete rules must match those generated by the ADD_SUBSET_OUTBOUND_RULES procedure. To view the rule names for subset rules, run the following query:

```
SELECT RULE_OWNER, SUBSETTING_OPERATION, RULE_NAME
   FROM DBA_XSTREAM_RULES
   WHERE SUBSETTING_OPERATION IS NOT NULL;
```

Note:

- This procedure removes the declarative rule-based transformation associated with each rule it removes.
- This procedure does not remove rules from the outbound server's capture process.

See Also: "ADD_SUBSET_OUTBOUND_RULES Procedure" on page 8-11

Syntax

```
DBMS_XSTREAM_ADM.REMOVE_SUBSET_OUTBOUND_RULES(
   server_name      IN VARCHAR2,
   insert_rule_name IN VARCHAR2,
   update_rule_name IN VARCHAR2,
   delete_rule_name IN VARCHAR2);
```

Parameters

Table 8–11 REMOVE_SUBSET_OUTBOUND_RULES Procedure Parameters

Parameter	Description
`server_name`	The name of the outbound server from which rules are being removed. Specify an existing outbound server. Do not specify an owner.
`insert_rule_name`	The name of the insert rule being removed, specified as [*schema_name*.]*rule_name*. For example, to specify a rule in the `hr` schema named `rule1`, enter `hr.rule1`. If the schema is not specified, then the current user is the default. If `NULL`, then the procedure raises an error.
`update_rule_name`	The name of the update rule being removed, specified as [*schema_name*.]*rule_name*. If `NULL`, then the procedure raises an error.
`delete_rule_name`	The name of the delete rule being removed, specified as [*schema_name*.]*rule_name*. If `NULL`, then the procedure raises an error.

9 DBMS_XSTREAM_AUTH

The `DBMS_XSTREAM_AUTH` package provides subprograms for granting privileges to and revoking privileges from XStream administrators.

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

This chapter contains the following topic:

See Also: "Granting Privileges for the XStream Administrator" on page 4-1

Using DBMS_XSTREAM_AUTH

This section contains topics which relate to using the `DBMS_XSTREAM_AUTH` package.

- Overview
- Security Model

Overview

This package provides subprograms for granting privileges to XStream administrators and revoking privileges from XStream administrators.

> **See Also:** "Granting Privileges for the XStream Administrator" on page 4-1

Security Model

Security on this package can be controlled in either of the following ways:

- Granting `EXECUTE` on this package to selected users or roles.
- Granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, connect as an administrative user who can create users, grant privileges, and create tablespaces when using this package.

Summary of DBMS_XSTREAM_AUTH Subprograms

Table 9–1 DBMS_XSTREAM_AUTH Package Subprograms

Subprogram	Description
GRANT_ADMIN_PRIVILEGE Procedure on page 9-6	Either grants the privileges needed by a user to be an XStream administrator directly, or generates a script that grants these privileges
GRANT_REMOTE_ADMIN_ACCESS Procedure on page 9-9	Enables a remote XStream administrator to perform administrative actions at the local database by connecting to the grantee using a database link
REVOKE_ADMIN_PRIVILEGE Procedure on page 9-10	Either revokes XStream administrator privileges from a user directly, or generates a script that revokes these privileges
REVOKE_REMOTE_ADMIN_ACCESS Procedure on page 9-12	Disables a remote XStream administrator from performing administrative actions by connecting to the grantee using a database link

Note: All subprograms commit unless specified otherwise.

GRANT_ADMIN_PRIVILEGE Procedure

This procedure either grants the privileges needed by a user to be an XStream administrator directly, or generates a script that grants these privileges.

See Also: "Granting Privileges for the XStream Administrator" on page 4-1

Syntax

```
DBMS_XSTREAM_AUTH.GRANT_ADMIN_PRIVILEGE(
   grantee            IN  VARCHAR2,
   grant_privileges   IN  BOOLEAN   DEFAULT TRUE,
   file_name          IN  VARCHAR2  DEFAULT NULL,
   directory_name     IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 9–2 GRANT_ADMIN_PRIVILEGE Procedure Parameters

Parameter	Description
`grantee`	The user to whom privileges are granted
`grant_privileges`	If `TRUE`, then the procedure grants the privileges to the specified grantee directly, and adds the grantee to the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view with `YES` for both the `LOCAL_PRIVILEGES` column and the `ACCESS_FROM_REMOTE` column. If the user already has an entry in this data dictionary view, then the procedure does not make another entry, and no error is raised. If `TRUE` and any of the grant statements fails, then the procedure raises an error. If `FALSE`, then the procedure does not grant the privileges to the specified grantee directly, and does not add the grantee to the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view. You specify `FALSE` when the procedure is generating a file that you will edit and then run. If you specify `FALSE` and either the `file_name` or `directory_name` parameter is `NULL`, then the procedure raises an error.
`file_name`	The name of the file generated by the procedure. The file contains all of the statements that grant the privileges. If a file with the specified file name exists in the specified directory name, then the grant statements are appended to the existing file. If `NULL`, then the procedure does not generate a file.
`directory_name`	The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement `CREATE DIRECTORY`. If you specify a directory, then the user who invokes the procedure must have the `WRITE` privilege on the directory object. If the `file_name` parameter is `NULL`, then this parameter is ignored, and the procedure does not generate a file. If `NULL` and the `file_name` parameter is non-`NULL`, then the procedure raises an error.

Usage Notes

The user who runs the procedure must be an administrative user who can grant privileges to other users.

Specifically, the procedure grants the following privileges to the specified user:

- The `RESTRICTED SESSION` system privilege
- `EXECUTE` on the following packages:
  - `DBMS_APPLY_ADM`
  - `DBMS_AQ`
  - `DBMS_AQADM`
  - `DBMS_AQIN`
  - `DBMS_AQELM`
  - `DBMS_CAPTURE_ADM`
  - `DBMS_FLASHBACK`
  - `DBMS_LOCK`
  - `DBMS_PROPAGATION_ADM`
  - `DBMS_RULE_ADM`
  - `DBMS_STREAMS_ADM`
  - `DBMS_STREAMS_ADVISOR_ADM`
  - `DBMS_STREAMS_HANDLER_ADM`
  - `DBMS_STREAMS_MESSAGING`
  - `DBMS_TRANSFORM`
  - `DBMS_XSTREAM_ADM`
- Privileges to enqueue messages into and dequeue messages from any queue
- Privileges to manage any queue
- Privileges to create, alter, and execute any of the following types of objects in the user's own schema and in other schemas:
  - Evaluation contexts
  - Rule sets
  - Rules

  In addition, the grantee can grant these privileges to other users.
- `SELECT_CATALOG_ROLE`
- `SELECT` privilege on data dictionary views related to XStream and Oracle Streams
- The ability to allow a remote XStream administrator to perform administrative actions through a database link by connecting to the grantee

  This ability is enabled by running the `GRANT_REMOTE_ADMIN_ACCESS` procedure in this package.

Note:

- To view all of the statements run by the procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.
- This procedure grants only the privileges necessary to configure and administer an XStream environment. You can grant additional privileges to the grantee if necessary.

See Also:

- "GRANT_REMOTE_ADMIN_ACCESS Procedure" on page 9-9
- "Granting Privileges for the XStream Administrator" on page 4-1
- *Oracle Database SQL Language Reference* for information about the `CREATE DIRECTORY` SQL statement

GRANT_REMOTE_ADMIN_ACCESS Procedure

This procedure enables a remote XStream administrator to perform administrative actions at the local database by connecting to the grantee using a database link.

Syntax

```
DBMS_XSTREAM_AUTH.GRANT_REMOTE_ADMIN_ACCESS(
   grantee  IN  VARCHAR2);
```

Parameters

Table 9–3 GRANT_REMOTE_ADMIN_ACCESS Procedure Parameter

Parameter	Description
`grantee`	The user who allows remote access. The procedure adds the grantee to the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view with `YES` for the `ACCESS_FROM_REMOTE` column. If the user already has an entry in this data dictionary view, then the procedure does not make another entry. Instead, it updates the `ACCESS_FROM_REMOTE` column to `YES`.

Usage Notes

Typically, you run the procedure and specify a grantee at a local source database if a downstream capture process captures changes originating at the local source database. The XStream administrator at a downstream capture database administers the source database using this connection.

Note: The `GRANT_ADMIN_PRIVILEGE` procedure in this package runs this procedure.

See Also: "GRANT_ADMIN_PRIVILEGE Procedure" on page 9-6

REVOKE_ADMIN_PRIVILEGE Procedure

This procedure either revokes XStream administrator privileges from a user directly, or generates a script that revokes these privileges.

Syntax

```
DBMS_XSTREAM_AUTH.REVOKE_ADMIN_PRIVILEGE(
   grantee            IN  VARCHAR2,
   revoke_privileges  IN  BOOLEAN   DEFAULT TRUE,
   file_name          IN  VARCHAR2  DEFAULT NULL,
   directory_name     IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 9–4 REVOKE_ADMIN_PRIVILEGE Procedure Parameters

Parameter	Description
`grantee`	The user from whom privileges are revoked
`revoke_privileges`	If `TRUE`, then the procedure revokes the privileges from the specified user directly, and removes the user from the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view. If the user does not have a record in this data dictionary view, then the procedure does not remove a record from the view, and no error is raised. If `TRUE` and any of the revoke statements fails, then the procedure raises an error. A revoke statement fails if the user is not granted the privilege that is being revoked. If `FALSE`, then the procedure does not revoke the privileges from the specified user directly, and does not remove the user from the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view. You specify `FALSE` when the procedure is generating a file that you will edit and then run. If you specify `FALSE` and either the `file_name` or `directory_name` parameter is `NULL`, then the procedure does not raise an error.
`file_name`	The name of the file generated by this procedure. The file contains all of the statements that revoke the privileges. If a file with the specified file name exists in the specified directory name, then the revoke statements are appended to the existing file. If `NULL`, then the procedure does not generate a file.
`directory_name`	The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement `CREATE DIRECTORY`. If you specify a directory, then the user who invokes the procedure must have the `WRITE` privilege on the directory object. If the `file_name` parameter is `NULL`, then this parameter is ignored, and the procedure does not generate a file. If `NULL` and the `file_name` parameter is non-`NULL`, then the procedure raises an error.

Usage Notes

The user who runs this procedure must be an administrative user who can revoke privileges from other users. Specifically, this procedure revokes the privileges granted by running the `GRANT_ADMIN_PRIVILEGE` procedure in this package.

Note: To view all of the statements run by this procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.

See Also:

- "GRANT_ADMIN_PRIVILEGE Procedure" on page 9-6
- *Oracle Database SQL Language Reference* for information about the `CREATE DIRECTORY` SQL statement

REVOKE_REMOTE_ADMIN_ACCESS Procedure

This procedure disables a remote XStream administrator from performing administrative actions by connecting to the grantee using a database link.

Note: The `REVOKE_ADMIN_PRIVILEGE` procedure in this package runs this procedure.

See Also: "REVOKE_ADMIN_PRIVILEGE Procedure" on page 9-10

Syntax

```
DBMS_XSTREAM_AUTH.REVOKE_REMOTE_ADMIN_ACCESS(
   grantee  IN  VARCHAR2);
```

Parameters

Table 9–5 REVOKE_REMOTE_ADMIN_ACCESS Procedure Parameter

Parameter	Description
`grantee`	The user for whom access from a remote XStream administrator is disabled. If a row for the grantee exists in the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view, then the procedure updates the `ACCESS_FROM_REMOTE` column for the grantee to `NO`. If, after this update, both the `LOCAL_PRIVILEGES` column and the `ACCESS_FROM_REMOTE` column are `NO` for the grantee, then the procedure removes the grantee from the view. If no row for the grantee exists in the `DBA_XSTREAM_ADMINISTRATOR` data dictionary view, then the procedure does not update the view and does not raise an error.

Part IV

XStream OCI API Reference

This part contains the XStream OCI API reference. This part contains the following chapters:

10 Introduction to the OCI Interface for XStream

The Oracle Call Interface (OCI) includes an interface for XStream. This chapter provides an introduction to the OCI interface for XStream.

This chapter contains these topics:

- About the XStream Interface
- Handler and Descriptor Attributes

This chapter provides an overview of the OCI interface for XStream. For detailed information about XStream concepts, see Chapter 2, "XStream Concepts".

See Also:

- Chapter 11, "OCI XStream Functions"
- Chapter 4, "Configuring XStream"
- Chapter 5, "Managing XStream"
- Chapter 6, "Monitoring XStream"
- Chapter 7, "Troubleshooting XStream"

About the XStream Interface

Since Oracle Database 11g Release 2, APIs, known as XStream Out and XStream In, are available. This technology enables high performance, near real-time information-sharing infrastructure between Oracle databases and non-Oracle databases, non-RDBMS Oracle products, file systems, third party software applications, and so on. XStream is built on top of Oracle Streams infrastructure.

See Also: Chapter 11, "OCI XStream Functions"

XStream Out

XStream Out allows a remote client to attach to an outbound server and extract row changes in the form of logical change records (LCRs). For the basics of LCRs, see *Oracle Streams Concepts and Administration*.

To use XStream Out, a capture process and an outbound server must be created. All data types supported by Oracle Streams, including LOB, `LONG`, and `XMLType`, are supported by XStream. The capture process and the outbound server need not be on the same database instance. After the capture process and the outbound server have started, row changes are captured and sent to the outbound server. An external client application can connect to this outbound server using OCI. After the connection is established, the client application can loop while waiting for LCRs from the outbound server. The client application can register a client-side callback to be invoked each time an LCR is received. At any time, the client application can detach from the outbound server as needed. Upon restart, the outbound server knows where in the redo stream to start streaming LCRs to the client application.

See Also: "XStream Out" on page 2-1

XStream In

To replicate non-Oracle data into Oracle databases, use XStream In. This technology allows a remote client application to attach to an inbound server and send row and DDL changes in the form of LCRs.

An external client application connects to the inbound server using OCI. After the connection is established, the client application acts as the capture agent for the inbound server by streaming LCRs to it. A client application can attach to only one inbound server for each database connection, and each inbound server only allows one client application to attach to it.

See Also: "XStream In" on page 2-9

Position Order and LCR Streams

Each LCR has a position attribute. The position of an LCR identifies its placement in the stream of LCRs in a transaction.

See Also: "Position Order in an LCR Stream" on page 2-11

XStream and Character Sets

XStream Out implicitly converts character data in LCRs from the outbound server database character set to the client application character set. XStream In implicitly converts character data in LCRs from the client application character set to the inbound server database character set.

To improve performance, complete the following tasks:

- Analyze the LCR data flow from the source to the destination.
- Set the client character set of the OCI client application to the one that minimizes character conversion, incurs no data loss, and takes advantage of the implicit conversion done by XStream or the destination.

For XStream Out, in general, setting the client application character set to the outbound server database character set is the best practice.

Handler and Descriptor Attributes

This chapter describes the attributes for OCI handles and descriptors, which can be read with `OCIAttrGet()` and modified with `OCIAttrSet()`.

Conventions

For each handle type, the attributes that can be read or changed are listed. Each attribute listing includes the following information:

Mode
The following modes are valid:

`READ` - The attribute can be read using `OCIAttrGet()`.

`WRITE` - The attribute can be modified using `OCIAttrSet()`.

`READ/WRITE` - The attribute can be read using `OCIAttrGet()`, and it can be modified using `OCIAttrSet()`.

Description
This is a description of the purpose of the attribute.

Attribute Data Type
This is the data type of the attribute. If necessary, a distinction is made between the data type for `READ` and `WRITE` modes.

Server Handle Attributes

The following server handle attributes are available:

- OCI_ATTR_XSTREAM_ACK_INTERVAL
- OCI_ATTR_XSTREAM_IDLE_TIMEOUT

OCI_ATTR_XSTREAM_ACK_INTERVAL

Mode
`READ/WRITE`

Description
For XStream Out, the ACK interval is the minimum interval in seconds that the outbound server receives the processed low position from the client application. After each ACK interval, the outbound server ends any in-progress `OCIXStreamOutLCRReceive()` or `OCIXStreamOutLCRCallbackReceive()` call so that the processed low position cached at the client application can be sent to the outbound server.

For XStream In, the ACK interval is the minimum interval in seconds that the inbound server sends the processed low position to the client application. After each ACK interval, any in-progress `OCIXStreamInLCRSend()` or `OCIXStreamInLCRCallbackSend()` call is terminated for the inbound server to send a new processed low position to the client application.

The default value for `OCI_ATTR_XSTREAM_ACK_INTERVAL` is 30 seconds. This attribute is checked only during the `OCIXStreamOutAttach()` or `OCIXStreamInAttach()` calls. Thus, it must be set before invoking these APIs; otherwise, the default value is used.

Attribute Data Type
`ub4 */ub4`

OCI_ATTR_XSTREAM_IDLE_TIMEOUT

Mode
`READ/WRITE`

Description
The idle timeout is the number of seconds of idle the outbound server waits for an LCR before terminating the `OCIXStreamOutLCRReceive()` or `OCIXStreamOutLCRCallbackReceive()` call.

The default for `OCI_ATTR_XSTREAM_IDLE_TIMEOUT` is one second. This attribute is checked only during the `OCIXStreamOutAttach()` or `OCIXStreamInAttach()` call. Thus, it must be set before invoking these APIs; otherwise, the default value is used.

Attribute Data Type

`ub4 */ub4`

11 OCI XStream Functions

This chapter describes the XStream functions for OCI.

A row logical change record (LCR) is used to encapsulate each row change. It includes the schema name, table name, DML operation, and the column values. For update operations, both before and after column values are included. The column data is in the format specified by the "Program Variable" column in Table 11–3. Character columns are converted to the client's character set.

A DDL LCR is used to encapsulate each DDL change. It includes the object name, the DDL text, and the DDL command, for example, `ALTER TABLE` or `TRUNCATE TABLE`. See *Oracle Call Interface Programmer's Guide* for a list of DDL command codes.

> **See Also:** *Oracle Database Globalization Support Guide* for more information about NLS settings.
>
> XStream sample programs are found in `xstream/oci` under the `$ORACLE_HOME/demo` directory.

Each LCR also has a transaction ID and position. For transactions captured outside Oracle databases, any byte-comparable `RAW` array can be used as the LCR position, if the position of each LCR in the stream is strictly increasing.

This chapter contains the topic:

- Introduction to XStream Functions
- OCI XStream Functions

Introduction to XStream Functions

This section includes the conventions used to describe the functions.

Conventions for OCI Functions

For each function, the following information is listed:

Purpose

A brief description of the action performed by the function.

Syntax

The function declaration.

Parameters

A description of each of the function's parameters. This includes the parameter's mode. The mode of a parameter has three possible values, as described in Table 11–1.

Table 11–1 Mode of a Parameter

Mode	Description
IN	A parameter that passes data to the OCI.
OUT	A parameter that receives data from the OCI on this call.
IN/OUT	A parameter that passes data on the call and receives data on the return from this or a subsequent call.

Comments

More detailed information about the function (if available), which can include return values, restrictions on the use of the function, examples, or other information that can be useful when using the function in an application.

OCI XStream Functions

This section and Table 11–1 describe the OCI XStream functions.

Table 11–2 OCI XStream Functions

Function	Purpose
LCR Functions	To get and set one or more values of an LCR. **Note:** These calls do not require a server round-trip.
"OCILCRAttributesGet()" on page 11-5	Returns existing extra attributes from the LCR
"OCILCRAttributesSet()" on page 11-7	Sets extra attributes in a row or DDL LCR
"OCILCRFree()" on page 11-9	Frees the LCR
"OCILCRHeaderGet()" on page 11-12	Returns the common header fields for a row/DDL LCR
"OCILCRHeaderSet()" on page 11-28	Initializes the common header fields for a row or DDL LCR
"OCILCRDDLInfoGet()" on page 11-10	Retrieves specific fields in a DDL LCR
"OCILCRDDLInfoSet()" on page 11-25	Populates DDL-specific fields in a DDL LCR
"OCILCRLobInfoGet()" on page 11-31	Returns the LOB information for a piece-wise LOB LCR
"OCILCRLobInfoSet()" on page 11-33	Sets the LOB information for a piece-wise LOB LCR
"OCILCRNew()" on page 11-18	Constructs a new LCR object of the specified type (ROW or DDL) for the given duration
"OCILCRRowColumnInfoGet()" on page 11-19	Returns the column fields in a row LCR
"OCILCRRowColumnInfoSet()" on page 11-22	Populates column fields in a row LCR
"OCILCRRowStmtGet()" on page 11-15	Returns the generated SQL statement for the row LCR, with values in-lined
"OCILCRRowStmtWithBindVarGet()" on page 11-16	Returns the generated SQL statement, which uses bind variables for column values
"OCILCRSCNsFromPosition()" on page 11-35	Gets the SCN and commit SCN from a position value
"OCILCRSCNToPosition()" on page 11-36	Converts SCN to position
"OCILCRWhereClauseGet()" on page 11-37	Gets the `WHERE` clause statement for the given row LCR
"OCILCRWhereClauseWithBindVarGet()" on page 11-39	Gets the `WHERE` clause statement with bind variables for the given row LCR
XStream In Functions	To send an LCR stream to an XStream inbound server
"OCIXStreamInAttach()" on page 11-41	Attaches to an inbound server
"OCIXStreamInChunkSend()" on page 11-55	Sends chunk data to the inbound server
"OCIXStreamInCommit()" on page 11-59	Commits the given transaction
"OCIXStreamInDetach()" on page 11-43	Detaches from the inbound server
"OCIXStreamInErrorGet()" on page 11-52	Returns the first error encountered by the inbound server since the attach call
"OCIXStreamInFlush()" on page 11-54	Flushes the network while attaching to an XStream inbound server
"OCIXStreamInLCRCallbackSend()" on page 11-46	Sends the LCR stream to the attached inbound server using callbacks

Table 11–2 (Cont.) OCI XStream Functions

Function	Purpose
"OCIXStreamInLCRSend()" on page 11-44	Sends the LCR stream to the attached inbound server using callbacks
"OCIXStreamInProcessedLWMGet()" on page 11-51	Gets the local processed low position
XStream Out Functions	To receive an LCR stream from an XStream outbound server
"OCIXStreamOutAttach()" on page 11-60	Attaches to an outbound server
"OCIXStreamOutChunkReceive()" on page 11-71	Retrieves data of each LOB or `LONG` or `XMLType` column one chunk at a time
"OCIXStreamOutDetach()" on page 11-62	Detaches from the outbound server
"OCIXStreamOutLCRCallbackReceive()" on page 11-65	Gets the LCR stream from the outbound server using callbacks
"OCIXStreamOutLCRReceive()" on page 11-63	Receives an LCR stream from an outbound server without using callbacks
"OCIXStreamOutProcessedLWMSet()" on page 11-70	Updates the local copy of the processed low-water mark

OCILCRAttributesGet()

Purpose

Gets extra attribute information in (ROW or DDL) LCR. In addition, it gets any extra non-first class attributes that are not populated through `OCILCRHeaderGet()`, `OCILCRDDLInfoGet()`, or `OCILCRRowColumnInfoGet()`, for example, edition name.

Syntax

```
sword OCILCRAttributesGet (      OCISvcCtx   *svchp,
                                 OCIError    *errhp,
                                 ub2         *num_attrs,
                                 oratext     **attr_names,
                                 ub2         *attr_namesl,
                                 ub2         *attr_dtyp,
                                 void        **attr_valuesp,
                                 OCIInd      *attr_indp,
                                 ub2         *attr_alensp,
                                 void        *lcrp,
                                 ub2         array_size,
                                 ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

num_attrs (OUT)
Number of extra attributes.

attr_names (OUT)
An array of extra attribute name pointers.

attr_namesl (OUT)
An array of extra attribute name lengths.

attr_dtyp (OUT)
An array of extra attribute data types. Valid data types: see Comments.

attr_valuesp (OUT)
An array of extra attribute data value pointers.

attr_indp (OUT)
An indicator array. Each returned element is an `OCIInd` value (`OCI_IND_NULL` or `OCI_IND_NOTNULL`).

attr_alensp (OUT)
An array of actual extra attribute data lengths. Each element in `alensp` is the length in bytes.

lcrp (IN)
Pointer to row or DDL LCR.

array_size (IN)
Size of the array argument in the other parameters. If `array_size` is not large enough to accommodate the number of attributes in the requested attribute list, then `OCI_ERROR` is returned. Parameter `num_attrs` returns the expected size.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

The valid data types for `attr_dtyp` are:

```
SQLT_CHR
SQLT_INT
SQLT_RDD
```

OCILCRAttributesSet()

Purpose

Populates extra attribute information in row or DDL LCR. In addition, it populates any extra non-first class attributes that cannot be set through `OCILCRHeaderSet()`, `OCILCRDDLInfoSet()`, or `OCILCRRowColumnInfoSet()`, for example, edition name.

Syntax

```
sword OCILCRAttributesSet (      OCISvcCtx   *svchp,
                                 OCIError    *errhp,
                                 ub2         num_attrs,
                                 oratext     **attr_names,
                                 ub2         *attr_names_lens,
                                 ub2         *attr_dtyp,
                                 void        **attr_valuesp,
                                 OCIInd      *attr_indp,
                                 ub2         *attr_alensp,
                                 void        *lcrp,
                                 ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

num_attrs (IN)
Number of extra attributes.

attr_names (IN)
Pointer to an array of extra attribute names. Attribute names must be canonicalized.

attr_names_lens (IN)
Pointer to an array of extra attribute name lengths.

attr_dtyp (IN)
Pointer to an array of extra attribute data types. See valid data types in Comments of "OCILCRRowColumnInfoSet()" on page 11-22.

attr_valuesp (IN)
Address of an array of extra attribute data values.

attr_indp (IN)
Pointer to an indicator array. For all data types, this is a pointer to an array of `OCIInd` values (`OCI_IND_NULL` or `OCI_IND_NOTNULL`).

attr_alensp (IN)
Pointer to an array of actual extra attribute data lengths. Each element in `attr_lensp` is the length in bytes.

lcrp (IN/OUT)
Pointer to a row or DDL LCR.

mode (IN)

Specify `OCI_DEFAULT`.

Comments

Valid attributes are:

```
#define OCI_LCR_ATTR_THREAD_NO              "THREAD#"
#define OCI_LCR_ATTR_ROW_ID                 "ROW_ID"
#define OCI_LCR_ATTR_SESSION_NO             "SESSION#"
#define OCI_LCR_ATTR_SERIAL_NO              "SERIAL#"
#define OCI_LCR_ATTR_USERNAME               "USERNAME"
#define OCI_LCR_ATTR_TX_NAME                "TX_NAME"
#define OCI_LCR_ATTR_EDITION_NAME           "EDITION_NAME"
#define OCI_LCR_ATTR_MESSAGE_TRACKING_LABEL "MESSAGE_TRACKING_LABEL"
```

OCILCRFree()

Purpose

Frees the LCR.

Syntax

```
sword OCILCRFree ( OCISvcCtx   *svchp,
                   OCIError    *errhp,
                   void        *lcrp,
                   ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

lcrp (IN/OUT)
Streams LCR pointer.

mode (IN)
Specify `OCI_DEFAULT`.

OCILCRDDLInfoGet()

Purpose

Retrieves specific fields in a DDL LCR.

Syntax

```
sword OCILCRDDLInfoGet ( OCISvcCtx   *svchp,
                         OCIError    *errhp,
                         oratext     **object_type,
                         ub2         *object_type_len,
                         oratext     **ddl_text,
                         ub4         *ddl_text_len,
                         oratext     **logon_user,
                         ub2         *logon_user_len,
                         oratext     **current_schema,
                         ub2         *current_schema_len,
                         oratext     **base_table_owner,
                         ub2         *base_table_owner_len,
                         oratext     **base_table_name,
                         ub2         *base_table_name_len,
                         oraub8      *flag,
                         void        *ddl_lcrp,
                         ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

object_type (OUT)
The type of object on which the DDL statement was executed. (See OCILCRDDLInfoSet().) Optional. If not `NULL`, then both `object_type` and `object_type_len` arguments must not be `NULL`.

object_type_len (OUT)
Length of the `object_type` string without the `NULL` terminator.

ddl_text (OUT)
The text of the DDL statement. Optional. If not `NULL`, then both `ddl_text` and `ddl_text_len` arguments must not be `NULL`.

ddl_text_len (OUT)
DDL text length in bytes without the `NULL` terminator.

logon_user (OUT)
Canonicalized (follows a rule or procedure) name of the user whose session executed the DDL statement. Optional. If not `NULL`, then both `logon_user` and `logon_user_len` arguments must not be `NULL`.

logon_user_len (OUT)
Length of the logon_user string without the `NULL` terminator.

current_schema (OUT)
The canonicalized schema name that is used if no schema is specified explicitly for the modified database objects in `ddl_text`. Optional. If not `NULL`, then both `current_schema` and `current_schema_len` arguments must not be `NULL`.

current_schema_len (OUT)
Length of the `current_schema` string without the `NULL` terminator.

base_table_owner (OUT)
If the DDL statement is a table-related DDL (such as `CREATE TABLE` and `ALTER TABLE`), or if the DDL statement involves a table (such as creating a trigger on a table), then `base_table_owner` specifies the canonicalized owner of the table involved. Otherwise, `base_table_owner` is `NULL`. Optional. If not `NULL`, then both `base_table_owner` and `base_table_owner_len` arguments must not be `NULL`.

base_table_owner_len (OUT)
Length of the `base_table_owner` string without the `NULL` terminator.

base_table_name (OUT)
If the DDL statement is a table-related DDL (such as `CREATE TABLE` and `ALTER TABLE`), or if the DDL statement involves a table (such as creating a trigger on a table), then `base_table_name` specifies the canonicalized name of the table involved. Otherwise, `base_table_name` is `NULL`. Optional. If not `NULL`, then both `base_table_name` and `base_table_name_len` arguments must not be `NULL`.

base_table_name_len (OUT)
Length of the `base_table_name` string without the `NULL` terminator.

flag (OUT)
DDL LCR flag. Optional. Data not returned if argument is `NULL`. Future extension not used currently.

ddl_lcrp (IN)
DDL LCR. Cannot be `NULL`.

mode (IN)
Specify `OCI_DEFAULT`.

OCILCRHeaderGet()

Purpose

Returns the common header fields for row or DDL LCR. All returned pointers point directly to the corresponding LCR fields.

Syntax

```
sword OCILCRHeaderGet ( OCISvcCtx   *svchp,
                        OCIError    *errhp,
                        oratext     **src_db_name,
                        ub2         *src_db_name_len,
                        oratext     **cmd_type,
                        ub2         *cmd_type_len,
                        oratext     **owner,
                        ub2         *owner_len,
                        oratext     **oname,
                        ub2         *oname_len,
                        ub1         **tag,
                        ub2         *tag_len,
                        oratext     **txid,
                        ub2         *txid_len,
                        OCIDate     *src_time,
                        ub2         *old_columns,
                        ub2         *new_columns,
                        ub1         **position,
                        ub2         *position_len,
                        oraub8      *flag,
                        void        *lcrp,
                        ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

src_db_name (OUT)
Canonicalized source database name. Must be non-`NULL`.

src_db_name_len (OUT)
Length of the `src_db_name` string in bytes excluding the `NULL` terminator.

cmd_type (OUT)
For row LCRs: One of the following values:

Note: The values, `#define OCI_LCR_ROW_CMD_ROLLBACK` and `#define OCI_LCR_ROW_CMD_START_TX`, is functionality that is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

```
#define OCI_LCR_ROW_CMD_INSERT
#define OCI_LCR_ROW_CMD_DELETE
#define OCI_LCR_ROW_CMD_UPDATE
```

```
#define OCI_LCR_ROW_CMD_COMMIT
#define OCI_LCR_ROW_CMD_ROLLBACK
#define OCI_LCR_ROW_CMD_START_TX
#define OCI_LCR_ROW_CMD_LOB_WRITE
#define OCI_LCR_ROW_CMD_LOB_TRIM
#define OCI_LCR_ROW_CMD_LOB_ERASE
```

For DDL LCRs: One of the command types in *Oracle Call Interface Programmer's Guide*.

cmd_type_len (OUT)
Length of the `cmd_type` string in bytes excluding the `NULL` terminator.

owner (OUT)
Canonicalized table owner name. Must be non-`NULL`.

owner_len (OUT)
Length of the `owner` string in bytes excluding the `NULL` terminator.

oname (OUT)
Canonicalized table name. Must be non-`NULL`

oname_len (OUT)
Length of the `oname` string in bytes excluding the `NULL` terminator.

tag (OUT)
A binary tag that enables tracking of the LCR. For example, you can use this tag to determine the original source database of the DML statement if apply forwarding is used.

tag_len (OUT)
Number of bytes in the tag.

txid (OUT)
Transaction ID. Must be non-`NULL`

txid_len (OUT)
Length of the string in bytes excluding the `NULL` terminator.

src_time (OUT)
The time when the change was generated in the redo log file of the source database.

old_columns (OUT)
Number of columns in the `OLD` column list. Returns 0 if the input LCR is a DDL LCR. Optional.

new_columns (OUT)
Number of columns in the `NEW` column list. Returns 0 if the input LCR is a DDL LCR. Optional.

position (OUT)
Position for LCR.

position_len (OUT)
Length of `position`.

flag (OUT)
LCR flag. Possible flags are listed in Comments.

lcrp (IN)

`lcrp` cannot be `NULL`.

mode (IN)

`OCILCR_NEW_ONLY_MODE` - If this mode is specified, then the `new_columns` returned is the count of the columns in the `NEW` column list only. Otherwise, the `new_columns` returned is the number of distinct columns present in either the `NEW` or the `OLD` column list of the given row LCR.

Comments

LCR flag.

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

```
#define OCI_ROWLCR_HAS_ID_KEY_ONLY  /* only has ID key cols */
#define OCI_ROWLCR_SEQ_LCR          /* sequence lcr */
```

OCILCRRowStmtGet()

Purpose

Returns the generated SQL statement for the row LCR, with values in-lined. Users must preallocate the memory for `sql_stmt`, and `*sql_stmt_len` must be set to the size of the allocated buffer, when it is passed in. If `*sql_stmt_len` is not large enough to hold the generated SQL statement, then an error is raised.

Syntax

```
sword OCILCRRowStmtGet ( OCISvcCtx   *svchp,
                         OCIError    *errhp,
                         oratext     *row_stmt,
                         ub4         *row_stmt_len,
                         void        *row_lcrp,
                         ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

row_stmt (IN/OUT)
The generated SQL statement for the row LCR.

row_stmt_len (IN/OUT)
Set to the size of the allocated buffer for `row_stmt` when passed in; returns the length of `row_stmt`.

row_lcrp (IN)
Pointer to row LCR.

mode (IN)
Specify `OCI_DEFAULT`.

OCILCRRowStmtWithBindVarGet()

Purpose

Returns the generated SQL statement, which uses bind variables for column values. The values for the bind variables are returned separately in arrays. You must preallocate the memory for `sql_stmt` and the arrays, `*sql_stmt_len` must be set to the size of the allocated buffer, and `array_size` must be the length of the arrays. The actual column values in `bind_var_valuesp` points to the values inside the LCR, so it is a shallow copy. If `array_size` is not large enough to hold all the variables, or if `*sql_stmt_len` is not large enough to hold the generated SQL statement, then an error is raised.

Syntax

```
sword OCILCRRowStmtWithBindVarGet ( OCISvcCtx    *svchp,
                                    OCIError     *errhp,
                                    oratext      *row_stmt,
                                    ub4          *row_stmt_len,
                                    ub2          *num_bind_var,
                                    ub2          *bind_var_dtyp,
                                    void         **bind_var_valuesp,
                                    OCIInd       *bind_var_indp,
                                    ub2          *bind_var_alensp,
                                    ub1          *bind_var_csetidp,
                                    ub1          *bind_var_csetfp,
                                    void         *row_lcrp,
                                    oratext      **chunk_column_names,
                                    ub2          *chunk_column_namesl,
                                    oraub8       *chunk_column_flags,
                                    ub2          array_size,
                                    oratext      *bind_var_syntax,
                                    ub4          mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

row_stmt (IN/OUT)
The generated SQL statement for the row LCR.

row_stmt_len (IN/OUT)
Set to the size of the allocated buffer for `row_stmt` when passed in; returns the length of `row_stmt`.

num_bind_var (OUT)
The number of bind variables.

bind_var_dtyp (IN/OUT)
Array of data types for the bind variables.

bind_var_valuesp (IN/OUT)
Array of values for the bind variables.

bind_var_indp (IN/OUT)
Array of `NULL` indicators for the bind variables.

bind_var_alensp (IN/OUT)
Array of lengths for the bind variable values.

bind_var_csetidp (IN/OUT)
Array of character set IDs for the bind variables.

bind_var_csetfp (IN/OUT)
Array of character set forms for the bind variables.

row_lcrp (IN)
Pointer to row LCR.

chunk_column_names (OUT)
Array of LOB column names in LCR.

chunk_column_namesl (OUT)
Array of LOB column name lengths.

chunk_column_flags (OUT)
Array of LOB column flags. Possible flags are listed in Comments.

array_size (IN)
Size of each of the parameter arrays.

bind_var_syntax (IN)
Either (`:`) (binds are of the form `:1`, `:2`, and so on.) or (`?`) (binds are of the form (`?`)).

mode (IN)
Specify `OCI_DEFAULT`.

Comments

The following LCR column flags can be combined using bitwise `OR` operator.

```
#define OCI_LCR_COLUMN_LOB_DATA     /* column contains LOB data */
#define OCI_LCR_COLUMN_LONG_DATA    /* column contains long data */
#define OCI_LCR_COLUMN_EMPTY_LOB    /* column has an empty LOB  */
#define OCI_LCR_COLUMN_LAST_CHUNK   /* last chunk of current column */
#define OCI_LCR_COLUMN_AL16UTF16    /* column is in AL16UTF16 fmt */
#define OCI_LCR_COLUMN_NCLOB        /* column has NCLOB data */
#define OCI_LCR_COLUMN_XML_DATA     /* column contains xml data */
#define OCI_LCR_COLUMN_XML_DIFF     /* column contains xmldiff data */
#define OCI_LCR_COLUMN_ENCRYPTED    /* column is encrypted */
#define OCI_LCR_COLUMN_UPDATED      /* col is updated */
/* OCI_LCR_COLUMN_UPDATED is set only for the modified columns in the NEW
 * column list of an update LCR.
 */
```

OCILCRNew()

Purpose

Constructs a new Streams LCR object of the specified type (ROW or DDL) for the given duration.

Syntax

```
sword OCILCRNew ( OCISvcCtx     *svchp,
                  OCIError      *errhp,
                  OCIDuration   duration,
                  ub1           lcrtype,
                  void          **lcrp,
                  ub4           mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

duration (IN)
Memory for the LCR is allocated for this specified duration.

lcrtype (IN)
LCR type. Values are:

```
#define OCI_LCR_XROW
#define OCI_LCR_XDDL
```

lcrp (IN/OUT)
If `*lcrp` is not `NULL`, an error is raised.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Note:

- After creation, you are not allowed to change the type of the LCR (ROW or DDL) or duration of the memory allocation.
- Use `OCILCRHeaderSet()` to populate common header fields for row or DDL LCR.
- After the LCR header is initialized, use `OCILCRRowColumnInfoSet()` or `OCILCRDDLInfoSet()` to populate operation specific elements. Use `OCILCRExtraAttributesSet()` to populate extra attribute information.
- Use `OCILCRFree()` to free the LCR created by this function.

OCILCRRowColumnInfoGet()

Purpose

Returns the column fields in a row LCR.

Syntax

```
sword OCILCRRowColumnInfoGet ( OCISvcCtx   *svchp,
                               OCIError    *errhp,
                               ub2         column_value_type,
                               ub2         *num_columns,
                               oratext     **column_names,
                               ub2         *column_name_lens,
                               ub2         *column_dtyp,
                               void        **column_valuesp,
                               OCIInd      *column_indp,
                               ub2         *column_alensp,
                               ub1         *column_csetfp,
                               oraub8      *column_flags,
                               ub2         *column_csid,
                               void        *row_lcrp,
                               ub2         array_size,
                               ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

column_value_type (IN)
ROW LCR column value type; either of:

```
#define OCI_LCR_ROW_COLVAL_OLD
#define OCI_LCR_ROW_COLVAL_NEW
```

num_columns (OUT)
Number of columns in the specified column array.

column_names (OUT)
An array of column name pointers.

column_name_lens (OUT)
An array of column name lengths.

column_dtyp (OUT)
An array of column data types. Optional. Data is not returned if `column_dtyp` is `NULL`.

column_valuesp (OUT)
An array of column data pointers.

column_indp (OUT)
An array of indicators.

column_alensp (OUT)
An array of column lengths. Each returned element is the length in bytes.

column_csetfp (OUT)
An array of character set forms for the columns. Optional. Data is not returned if the argument is `NULL`.

column_flags (OUT)
An array of column flags. Optional. Data is not returned if the argument is `NULL`. See Comments for the values.

column_csid (OUT)
An array of character set IDs for the columns.

row_lcrp (IN)
`row_lcrp` cannot be `NULL`.

array_size (IN)
Size of each of the parameter arrays. An error is returned if `array_size` is less than the number of columns in the requested column list. The actual size of the requested column list is returned through the `num_columns` parameter.

mode (IN)
`OCILCR_NEW_ONLY_MODE` - If this mode is specified, then the `new_columns` returned is the count of the columns in the `NEW` column list only. Otherwise, the `new_columns` returned is the number of distinct columns present in either the `NEW` or the `OLD` column list of the given row LCR.

Comments

- For `INSERT`, this function must only be called to get the NEW column values.
- For `DELETE`, this function must only be called to get the OLD column values.
- For `UPDATE`, this function can be called twice, once to get the NEW column values and once to get the OLD column values.
- This function must not be called for `COMMIT` operations.

The following LCR column flags can be combined using bitwise `OR` operator.

```
#define OCI_LCR_COLUMN_LOB_DATA     /* column contains LOB data */
#define OCI_LCR_COLUMN_LONG_DATA    /* column contains long data */
#define OCI_LCR_COLUMN_EMPTY_LOB    /* column has an empty LOB  */
#define OCI_LCR_COLUMN_LAST_CHUNK   /* last chunk of current column */
#define OCI_LCR_COLUMN_AL16UTF16    /* column is in AL16UTF16 fmt */
#define OCI_LCR_COLUMN_NCLOB        /* column has NCLOB data */
#define OCI_LCR_COLUMN_XML_DATA     /* column contains xml data */
#define OCI_LCR_COLUMN_XML_DIFF     /* column contains xmldiff data */
#define OCI_LCR_COLUMN_ENCRYPTED    /* column is encrypted */
#define OCI_LCR_COLUMN_UPDATED      /* col is updated */
/* OCI_LCR_COLUMN_UPDATED is set only for the modified columns in the NEW
 * column list of an update LCR.
 */
```

Table 11–3 lists the currently supported table column data types. For each data type, it lists the corresponding LCR column data type, the C program variable type to cast the LCR column value, and the OCI functions that can manipulate the column values returned from `OCILCRRowColumnInfoGet()`.

Table 11–3 Table Column Data Types

Table Column Data Types	LCR Column Data Type	Program Variable	Conversion Function
VARCHAR, NVARCHAR2	SQLT_CHR	char *	

NUMBER	SQLT_VNU	OCINumber	OCINumberToInt()
			OCINumberToReal()
			OCINumberToText()

DATE	SQLT_ODT	OCIDate	OCIDateToText() Can access structure directly to get date and time fields.
RAW	SQLT_BIN	unsigned char *	
CHAR, NCHAR	SQLT_AFC	char *	
BINARY_FLOAT	SQLT_BFLOAT	float	
BINARY_DOUBLE	SQLT_BDOUBLE	double	

TIMESTAMP	SQLT_TIMESTAMP	OCIDateTime *	OCIDateTimeGetTime()
			OCIDateTimeGetDate()
			OCIDateTimeGetTimeZoneOffset()
			OCIDateTimeToText()

TIMESTAMP WITH TIME ZONE	SQLT_TIMESTAMP_TZ	OCIDateTime *	OCIDateTimeGetTime()
			OCIDateTimeGetDate()
			OCIDateTimeGetTimeZoneOffset()
			OCIDateTimeToText()

TIMESTAMP WITH LOCAL TIME ZONE	SQLT_TIMESTAMP_LTZ	OCIDateTime *	OCIDateTimeGetTime()
			OCIDateTimeGetDate()
			OCIDateTimeGetTimeZoneOffset()
			OCIDateTimeToText()

INTERVAL YEAR TO MONTH	SQLT_INTERVAL_YM	OCIInterval *	OCIIntervalToText()
			OCIIntervalGetYearMonth()

INTERVAL DAY TO SECOND	SQLT_INTERVAL_DS	OCIInterval *	OCIIntervalToText()
			OCIIntervalGetDaySecond()

UROWID	SQLT_RDD	OCIRowid *	OCIRowidToChar()
CLOB	SQLT_CHR or SQLT_BIN	unsigned char *	*
NCLOB	SQLT_BIN	unsigned char *	*
BLOB	SQLT_BIN	unsigned char *	*
LONG	SQLT_CHR	char *	*
LONG RAW	SQLT_BIN	unsigned char *	*

* Call OCIXStreamOutChunkReceive() to get column data.

OCILCRRowColumnInfoSet()

Purpose

Populates column fields in a row LCR.

Syntax

```
sword OCILCRRowColumnInfoSet ( OCISvcCtx   *svchp,
                               OCIError    *errhp,
                               ub2         column_value_type,
                               ub2         num_columns,
                               oratext     **column_names,
                               ub2         *column_name_lens,
                               ub2         *column_dtyp,
                               void        **column_valuesp,
                               OCIInd      *column_indp,
                               ub2         *column_alensp,
                               ub1         *column_csetfp,
                               oraub8      *column_flags,
                               ub2         *column_csid,
                               void        *row_lcrp,
                               ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

column_value_type (IN)
ROW LCR Column value types:

```
#define OCI_LCR_ROW_COLVAL_OLD
#define OCI_LCR_ROW_COLVAL_NEW
```

num_columns (IN)
Number of columns in each of the array parameters.

column_names (IN)
Pointer to an array of column names. Column names must be canonicalized. Column names must follow Oracle Database naming conventions and size limitations.

column_name_lens (IN)
Pointer to an array of column name lengths.

column_dtyp (IN)
Pointer to an array of column data types. See Comments for valid data types.

column_valuesp (IN)
Pointer to an array of column data pointers.

column_indp (IN)
Pointer to an indicator array. For all data types, this is a pointer to an array of `OCIInd` values (`OCI_IND_NULL` or `OCI_IND_NOTNULL`).

column_alensp (IN)
Pointer to an array of actual column lengths in bytes.

column_csetfp (IN)
Pointer to an array of character set forms for the columns. The default form is `SQLCS_IMPLICIT`. Setting this attribute causes the database or national character set to be used on the client side. Set this attribute to `SQLCS_NCHAR` for the national character set or `SQLCS_IMPLICIT` for the database character set. Pass 0 for non-character columns.

column_flags (IN)
Pointer to an array of column flags. (See Comments for the list of valid LCR column flags.)

column_csid (IN)
Pointer to an array of character set IDs for the columns.

row_lcrp (IN/OUT)
`row_lcrp` cannot be `NULL`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Note:

- For `INSERT`, this function must only be called to specify the NEW column values.
- For `DELETE`, this function must only be called to specify the OLD column values.
- For `UPDATE`, this function can be called twice, once to specify the NEW column values and once to specify the OLD column values.
- This function must not be called for `COMMIT` operations.

The following LCR column flags can be combined using the bitwise `OR` operator.

```
#define OCI_LCR_COLUMN_LOB_DATA     /* column contains LOB data */
#define OCI_LCR_COLUMN_LONG_DATA    /* column contains long data */
#define OCI_LCR_COLUMN_EMPTY_LOB    /* column has an empty LOB  */
#define OCI_LCR_COLUMN_LAST_CHUNK   /* last chunk of current column */
#define OCI_LCR_COLUMN_AL16UTF16    /* column is in AL16UTF16 fmt */
#define OCI_LCR_COLUMN_NCLOB        /* column has NCLOB data */
#define OCI_LCR_COLUMN_XML_DATA     /* column contains xml data */
#define OCI_LCR_COLUMN_XML_DIFF     /* column contains xmldiff data */
#define OCI_LCR_COLUMN_ENCRYPTED    /* column is encrypted */
#define OCI_LCR_COLUMN_UPDATED      /* col is updated */
/* OCI_LCR_COLUMN_UPDATED is set only for the modified columns in the NEW
 * column list of an update LCR.
 */
```

Valid data types are:

```
SQLT_AFC           SQLT_TIMESTAMP
SQLT_DAT           SQLT_TIMESTAMP_TZ
SQLT_BFLOAT        SQLT_TIMESTAMP_LTZ
SQLT_BDOUBLE       SQLT_INTERVAL_YM
SQLT_NUM           SQLT_INTERVAL_DS
SQLT_VCS
SQLT_ODT
SQLT_INT
SQLT_BIN
```

```
SQLT_CHR
SQLT_RDD
SQLT_VST
SQLT_INT
SQLT_FLT
```

OCILCRDDLInfoSet()

Purpose

Populates DDL-specific fields in a DDL LCR.

Syntax

```
sword OCILCRDDLInfoSet ( OCISvcCtx    *svchp,
                         OCIError     *errhp,
                         oratext      *object_type,
                         ub2          object_type_len,
                         oratext      *ddl_text,
                         ub4          ddl_text_len,
                         oratext      *logon_user,
                         ub2          logon_user_len,
                         oratext      *current_schema,
                         ub2          current_schema_len,
                         oratext      *base_table_owner,
                         ub2          base_table_owner_len,
                         oratext      *base_table_name,
                         ub2          base_table_name_len,
                         oraub8       flag,
                         void         *ddl_lcrp,
                         ub4          mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

object_type (IN)
The type of object on which the DDL statement was executed. See Comments for the valid object types.

object_type_len (IN)
Length of the `object_type` string without the `NULL` terminator.

ddl_text (IN)
The text of the DDL statement. This parameter must be set to a non-`NULL` value. DDL text must be in Oracle Database DDL format.

ddl_text_len (IN)
DDL text length in bytes without the `NULL` terminator.

logon_user (IN)
Canonicalized name of the user whose session executed the DDL statement.

logon_user_len (IN)
Length of the `logon_user` string without the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

current_schema (IN)
The canonicalized schema name that is used if no schema is specified explicitly for the modified database objects in `ddl_text`. If a schema is specified in `ddl_text` that differs from the one specified for `current_schema`, then the function uses the schema specified in `ddl_text`.

This parameter must be set to a non-`NULL` value.

current_schema_len (IN)
Length of the `current_schema` string without the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

base_table_owner (IN)
If the DDL statement is a table-related DDL (such as `CREATE TABLE` or `ALTER TABLE`), or if the DDL statement involves a table (such as creating a trigger on a table), then `base_table_owner` specifies the canonicalized owner of the table involved. Otherwise, `base_table_owner` is `NULL`.

base_table_owner_len (IN)
Length of the `base_table_owner` string without the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

base_table_name (IN)
If the DDL statement is a table-related DDL (such as `CREATE TABLE` or `ALTER TABLE`), or if the DDL statement involves a table (such as creating a trigger on a table), then `base_table_name` specifies the canonicalized name of the table involved. Otherwise, `base_table_name` is `NULL`.

base_table_name_len (IN)
Length of the `base_table_name` without the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

flag (IN)
DDL LCR flag. (Not currently used; used for future extension.) Specify `OCI_DEFAULT`.

ddl_lcrp (IN/OUT)
`ddl_lcrp` cannot be `NULL`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

The following are valid object types:

```
CLUSTER
FUNCTION
INDEX
OUTLINE
PACKAGE
PACKAGE BODY
PROCEDURE
SEQUENCE
SYNONYM
TABLE
TRIGGER
TYPE
USER
VIEW
```

`NULL` is also a valid object type. Specify `NULL` for all object types not listed.

OCILCRHeaderSet()

Purpose

Initializes the common header fields for row or DDL LCR.

Syntax

```
sword OCILCRHeaderSet ( OCISvcCtx  *svchp,
                        OCIError   *errhp,
                        oratext    *src_db_name,
                        ub2        src_db_name_len,
                        oratext    *cmd_type,
                        ub2        cmd_type_len,
                        oratext    *owner,
                        ub2        owner_len,
                        oratext    *oname,
                        ub2        oname_len,
                        ub1        *tag,
                        ub2        tag_len,
                        oratext    *txid,
                        ub2        txid_len,
                        OCIDate    *src_time,
                        ub1        *position,
                        ub2        position_len,
                        oraub8     flag,
                        void       *lcrp,
                        ub4        mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

src_db_name (IN)
Canonicalized source database name. Must be non-`NULL`.

src_db_name_len (IN)
Length of the `src_db_name` string in bytes excluding the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

cmd_type (IN)
For row LCRs: One of the following values:

Note: The values, `#define OCI_LCR_ROW_CMD_ROLLBACK` and `#define OCI_LCR_ROW_CMD_START_TX`, are available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

```
#define OCI_LCR_ROW_CMD_INSERT
#define OCI_LCR_ROW_CMD_DELETE
#define OCI_LCR_ROW_CMD_UPDATE
#define OCI_LCR_ROW_CMD_COMMIT
#define OCI_LCR_ROW_CMD_ROLLBACK
```

```
#define OCI_LCR_ROW_CMD_START_TX
#define OCI_LCR_ROW_CMD_LOB_WRITE
#define OCI_LCR_ROW_CMD_LOB_TRIM
#define OCI_LCR_ROW_CMD_LOB_ERASE
```

For DDL LCRs: One of the command types in *Oracle Call Interface Programmer's Guide*.

cmd_type_len (IN)
Length of `cmd_type`.

owner (IN)
Canonicalized table owner name. Owner is not required for `COMMIT` LCR.

owner_len (IN)
Length of the `owner` string in bytes excluding the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

oname (IN)
Canonicalized table name. Owner is not required for `COMMIT` LCR.

oname_len (IN)
Length of the `oname` string in bytes excluding the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

tag (IN)
A binary tag that enables tracking of the LCR. For example, you can use this tag to determine the original source database of the DML statement if apply forwarding is used.

tag_len (IN)
Number of bytes in the tag. Cannot exceed 2000 bytes.

txid (IN)
Transaction ID. Must be non-`NULL`.

txid_len (IN)
Length of the `txid` string in bytes, excluding the `NULL` terminator. Must follow Oracle Database naming conventions and size limitations.

src_time (IN)
The time when the change was generated in the online redo log file of the source database.

position (IN)
Position for LCR. Must be non-`NULL` and byte-comparable.

position_len (IN)
Length of position. Must be greater than zero.

flag (IN)
LCR flag. Possible flags are listed in Comments.

lcrp (IN/OUT)
`lcrp` cannot be `NULL`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Note:

- This function sets all internal fields of the LCR to `NULL` including extra attributes.
- This function *does not* deep copy the passed-in values. You must ensure data is valid for the duration of the LCR.
- For `COMMIT` LCRs, `owner` and `oname` information are not required. Provide valid values for `src_db_name`, `cmd_type`, `tag`, `txid`, and `position`.
- For row LCRs, use `OCILCRRowColumnInfoSet()` to populate row LCR-specific column information.
- For DDL LCRs, use `OCILCRDDLInfoSet()` to populate DDL operation specific information.
- For row or DDL LCRs, use `OCILCRAttributesSet()` to populate extra attribute information.

The following are LCR flags:

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

```
#define OCI_ROWLCR_HAS_ID_KEY_ONLY  /* only has ID key cols */
#define OCI_ROWLCR_SEQ_LCR          /* sequence lcr */
```

OCILCRLobInfoGet()

Purpose

Returns the LOB information for a piece-wise LOB LCR generated from a `DBMS_LOB` or `OCILob` procedure.

Syntax

```
sword OCILCRLobInfoGet ( OCISvcCtx   *svchp,
                         OCIError    *errhp,
                         oratext     **column_name,
                         ub2         *column_name_len,
                         ub2         *column_dty,
                         oraub8      *column_flag,
                         ub4         *offset,
                         ub4         *size,
                         void        *row_lcrp,
                         ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

column_name (OUT)
LOB column name.

column_name_len (OUT)
Length of LOB column name.

column_dty (OUT)
Column data type (either `SQLT_CHR` or `SQLT_BIN`).

column_flag (OUT)
Column flag. See Comments in "OCILCRRowColumnInfoSet()" on page 11-22.

offset (OUT)
LOB operation offset in code points. Only returned for `LOB WRITE` and `LOB TRIM` operations. This is the same as the `offset` parameter for `OCILobErase()` or the `offset` parameter in `OCILobWrite()`.

size (OUT)
LOB operation size in code points. Only returned for `LOB TRIM` and `LOB ERASE` operations. This is the same as the `new_length` parameter in `OCILobTrim()` or the `amtp` parameter in `OCILobErase()`.

row_lcrp (IN)
Pointer to a row LCR.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Returns `OCI_SUCCESS` or `OCI_ERROR`.

OCILCRLobInfoSet()

Purpose

Sets the LOB information for a piece-wise LOB LCR. This call is valid when the input LCR is a `LOB_WRITE`, `LOB_ERASE`, or `LOB_TRIM`; otherwise, an error is returned.

Syntax

```
sword OCILCRLobInfoSet ( OCISvcCtx   *svchp,
                         OCIError    *errhp,
                         oratext     *column_name,
                         ub2         column_name_len,
                         ub2         column_dty,
                         oraub8      column_flag,
                         ub4         offset,
                         ub4         size,
                         void        *row_lcrp,
                         ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

column_name (IN)
LOB column name.

column_name_len (IN)
Length of LOB column name.

column_dty (IN)
Column data type (either `SQLT_CHR` or `SQLT_BIN`).

column_flag (IN)
Column flag. See Comments in "OCILCRRowColumnInfoSet()" on page 11-22.

offset (IN)
LOB operation offset in code points. Only required for `LOB WRITE` and `LOB TRIM` operations. This is the same as the `soffset` parameter for `OCILobErase()` or the `offset` parameter in `OCILobWrite()`.

size (IN)
LOB operation size in code points. Only required for `LOB TRIM` and `LOB ERASE` operations.This is the same as the `new_length` parameter in `OCILobTrim()` or the `amtp` parameter in `OCILobErase()`.

row_lcrp (IN/OUT)
Pointer to a row LCR.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Returns `OCI_SUCCESS` or `OCI_ERROR`.

OCILCRSCNsFromPosition()

Purpose

Returns the SCN and the commit SCN from the position value. The input position must be one that is obtained from an XStream outbound server. An error is returned if the input position does not conform to the expected format.

Syntax

```
sword OCILCRSCNsFromPosition ( OCISvcCtx   *svchp,
                               OCIError    *errhp,
                               ub1         *position,
                               ub2         position_len,
                               OCINumber   *scn,
                               OCINumber   *commit_scn,
                               ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

position (IN)
LCR position value.

position_len (IN)
Length of LCR position value.

scn (OUT)
SCN number embedded in the given LCR position.

commit_scn (OUT)
The commit SCN embedded in the given position.

mode (IN)
Mode flags used for future expansion. Specify `OCI_DEFAULT`.

OCILCRSCNToPosition()

Purpose

Converts an SCN to a position. The generated position can be passed as the `last_position` to `OCIXStreamOutAttach()` to filter the LCRs with commit SCN less than the given SCN and the LCR's SCN less than the given SCN. Therefore, the first LCR sent by the outbound server is either:

- A commit LCR at the given SCN, or
- The first LCR of the subsequent transaction with commit SCN greater than or equal to the given SCN.

Syntax

```
sword OCILCRSCNToPosition ( OCISvcCtx  *svchp,
                            OCIError   *errhp,
                            ub1        *position,
                            ub2        *position_len,
                            OCINumber  *scn,
                            ub4        mode );
```

Parameters

svchp (IN)
OCI service context.

errhp (IN)
OCI error handle.

position (OUT)
The resulting position. You must preallocate `OCI_LCR_MAX_POSITION_LEN` bytes.

position_len (OUT)
Length of `position`.

scn (IN)
The SCN to be stored in `position`.

mode (IN)
Mode flags (Not currently used; used for future extension).

Comments

Returns `OCI_SUCCESS` if the conversion succeeds, `OCI_ERROR` otherwise.

OCILCRWhereClauseGet()

Purpose

Gets the WHERE clause statement for the given row LCR.

Syntax

```
sword OCILCRWhereClauseGet ( OCISvcCtx  *svchp,
                             OCIError   *errhp,
                             oratext    *wc_stmt,
                             ub4        *wc_stmt_len,
                             void       *row_lcrp,
                             ub4        mode );
```

Parameters

svchp (IN/OUT)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to OCIErrorGet() for diagnostic information in case of an error.

wc_stmt (OUT)
SQL statement equivalent to the LCR.

wc_stmt_len (IN/OUT)
Length of the wc_stmt buffer.

row_lcrp (IN)
Row LCR to be converted to SQL.

mode (IN)
Mode flags used for future expansion. Specify OCI_DEFAULT.

Comments

The WHERE clause generated for an INSERT LCR has all the columns that are being inserted. This WHERE clause could be used to identify the inserted row after it is inserted, for example, like "returning ROWID".

```
INSERT INTO TAB(COL1) VALUES (10) -> WHERE COL1=10
```

The WHERE clause generated for UPDATE has all the columns in the old column list. However, the values of the columns are that of the new value if it exists in the new column list of the UPDATE. If the column does not have a new value, then the old column value is used.

```
UPDATE TAB SET COL1 = 10 WHERE COL1 = 20 -> WHERE COL1 = 10
UPDATE TAB SET COL2 = 20 WHERE COL1 = 20 -> WHERE COL1 = 20
```

The WHERE clause for DELETE uses the columns and values from the old column list.

LOB piecewise operations use the new columns and values for generating the WHERE clause.

Returns

`OCI_SUCCESS` or `OCI_ERROR`.

OCILCRWhereClauseWithBindVarGet()

Purpose

Gets the `WHERE` clause statement with bind variables for the given row LCR.

Syntax

```
sword OCILCRWhereClauseWithBindVarGet ( OCISvcCtx  *svchp,
                                        OCIError   *errhp,
                                        oratext    *wc_stmt,
                                        ub4        *wc_stmt_len,
                                        ub2        *num_bind_var,
                                        ub2        *bind_var_dtyp,
                                        void       **bind_var_valuesp,
                                        OCIInd     *bind_var_indp,
                                        ub2        *bind_var_alensp,
                                        ub2        *bind_var_csetidp,
                                        ub1        *bind_var_csetfp,
                                        void       *row_lcrp,
                                        ub2        array_size,
                                        oratext    *bind_var_syntax,
                                        ub4        mode );
```

Parameters

svchp (IN/OUT)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

wc_stmt (OUT)
SQL statement equivalent to the LCR.

wc_stmt_len (IN/OUT)
Length of the `wc_stmt` buffer.

num_bind_var (OUT)
Number of bind variables.

bind_var_dtyp (OUT)
Array of data types of bind variables.

bind_var_valuesp (OUT)
Array of values of bind variables.

bind_var_indp (OUT)
Array of `NULL` indicators of bind variables.

bind_var_alensp (OUT)
Array of lengths of bind values.

bind_var_csetidp (OUT)
Array of char set IDs of binds.

bind_var_csetfp (OUT)
Array of char set forms of binds.

row_lcrp (IN)
Row LCR to be converted to SQL.

array_size (IN)
Size of the array of bind values.

bind_var_syntax (IN)
Native syntax to be used for binds.

mode (IN)
Mode flags for future expansion. Specify `OCI_DEFAULT`.

Comments

If `array_size` is not large enough to accommodate the number of columns in the requested column list, then `OCI_ERROR` is returned. The expected `array_size` is returned through the `num_bind_var` parameter.

`bind_var_syntax` for Oracle Database should contain (`:`). This generates positional binds such as `:1`, `:2`, `:3`, and so on. For non-Oracle databases input the string that must be used for binds.

The `WHERE` clause generated for `INSERT` LCR has all the columns that are being inserted. This `WHERE` clause can identify the inserted row after it is inserted, for example, like "returning `ROWID`".

```
INSERT INTO TAB(COL1) VALUES (10) -> WHERE COL1=10
```

The `WHERE` clause generated for `UPDATE` has all the columns in the old column list. However, the values of the columns are that of the new column value of the column if it exists in the new values of the `UPDATE`. If the column appears only in the old column, then the old column value is used.

```
UPDATE TAB SET COL1 = 10 WHERE COL1 = 20 -> WHERE COL1 = 10
UPDATE TAB SET COL2 = 20 WHERE COL1 = 20 -> WHERE COL1 = 20
```

The `WHERE` clause for `DELETE` uses the columns and values from the old column list.

LOB piecewise operations use the new columns and values for generating the `WHERE` clause.

Returns

`OCI_SUCCESS` or `OCI_ERROR`.

OCIXStreamInAttach()

Purpose

Attaches to an inbound server. The client application must connect to the database using a dedicated connection.

Syntax

```
sword OCIXStreamInAttach ( OCISvcCtx  *svchp,
                           OCIError   *errhp,
                           oratext    *server_name,
                           ub2        server_name_len,
                           oratext    *source_name,
                           ub2        source_name_len,
                           ub1        *last_position,
                           ub2        *last_position_len,
                           ub4        mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

server_name (IN)
XStream inbound server name.

server_name_len (IN)
Length of the XStream inbound server name.

source_name (IN)
Source name to identify the data source.

source_name_len (IN)
Source name length.

last_position (OUT)
Last position received by inbound server. Optional. If specified, then you must preallocate `OCI_LCR_MAX_POSITION_LEN` bytes for the return value.

last_position_len (OUT)
Length of `last_position`. Must be non-`NULL` if `last_position` is non-`NULL`.

mode (IN)

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

`OCIXSTREAM_IN_ATTACH_RESTART_INBOUND` - If this mode is specified, then this function can notify the server to restart the inbound server regardless of whether it is in a disabled or aborted state. If you do not pass in this mode and the inbound server is in an aborted state when this call is made, then the function returns an error.

Comments

The name of the inbound server must be provided because multiple inbound servers can be configured in one Oracle instance. This function returns `OCI_ERROR` if any error is encountered while attaching to the inbound server. Only one client can attach to an XStream inbound server at any time. An error is returned if multiple clients attempt to attach to the same inbound server or if the same client attempts to attach to multiple inbound servers concurrently.

This function returns the last position received by the inbound server. Having successfully attached to the server, the client should resume sending LCRs with positions greater than this `last_position` since the inbound server discards all LCRs with positions less than or equal to the `last_position`.

Returns either `OCI_SUCCESS` or `OCI_ERROR` status code.

OCIXStreamInDetach()

Purpose

Detaches from the inbound server.

Syntax

```
sword OCIXStreamInDetach ( OCISvcCtx  *svchp,
                           OCIError   *errhp,
                           ub1        *processed_low_position,
                           ub2        *processed_low_position_len,
                           ub4        mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

processed_low_position (OUT)
The server's processed low position.

processed_low_position (OUT)
Length of `processed_low_position`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

You must pass in a preallocated buffer for the position argument. The maximum length of this buffer is `OCI_LCR_MAX_POSITION_LEN`. This position is exposed in `DBA_XSTREAM_INBOUND_PROGRESS` view

This call returns the server's processed low position. If this function is invoked while a `OCIXStreamInLCRSend()` call is in progress, then it immediately terminates that call before detaching from the inbound server.

Returns either `OCI_SUCCESS` or `OCI_ERROR` status code.

OCIXStreamInLCRSend()

Purpose

Sends an LCR stream from the client to the attached inbound server. To avoid a network round-trip for every `OCIXStreamInLCRSend()` call, the connection is tied to this call and terminates the call after an ACK interval since the LCR stream is initiated to the server.

Syntax

```
sword OCIXStreamInLCRSend ( OCISvcCtx   *svchp,
                            OCIError    *errhp,
                            void        *lcrp,
                            ub1         lcrtype,
                            oraub8      flag,
                            ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

lcrp (IN)
Pointer to the new LCR to send. It cannot be `NULL`.

lcrtype (IN)
LCR type. Either of:

```
#define OCI_LCR_XROW
#define OCI_LCR_XDDL
```

flag (IN)
If bit `OCI_XSTREAM_MORE_ROW_DATA` (0x01) is set, then LCR contains more chunk data. You must call `OCIXStreamInChunkSend()` before calling `OCIXStreamInLCRSend()` again.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Return codes are:

- `OCI_STILL_EXECUTING` means that the current call is still in progress. The connection associated with the specified service context handle is still tied to this call for streaming the LCRs to the server. An error is returned if you attempt to use the same connection to execute any OCI calls that require database round-trip, for example, `OCIStmtExecute()`, `OCIStmtFetch()`, `OCILobRead()`, and so on. `OCILCR*` calls are local calls; thus, they are valid while this call is in progress.
- `OCI_SUCCESS` means the current call is completed. You can execute `OCIStmt*`, `OCILob*`, and so on from the same service context.
- `OCI_ERROR` means this call encounters some errors. Use `OCIErrorGet()` to obtain information about the error.

See Also: "Server Handle Attributes" on page 10-3

OCIXStreamInLCRCallbackSend()

Purpose

Sends an LCR stream to the attached inbound server. You must specify a callback to construct each LCR for streaming. If some LCRs contain chunk data, then a second callback must be provided to create each chunk data.

Syntax

```
sword OCIXStreamInLCRCallbackSend (
         OCISvcCtx                        *svchp,
         OCIError                         *errhp,
         OCICallbackXStreamInLCRCreate    createlcr_cb,
         OCICallbackXStreamInChunkCreate  createchunk_cb,
         void                             *usrctxp,
         ub4                              mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

createlcr_cb (IN)
Client callback procedure to be invoked to generate an LCR for streaming. Cannot be `NULL`.

createchunk_cb (IN)
Client callback procedure to be invoked to create each chunk. Can be `NULL` if you do not need to send any LCR with LOB or `LONG` or `XMLType` columns. `OCI_ERROR` is returned if this argument is `NULL` and you attempt to send an LCR with additional chunk data.

usrctxp (IN)
User context to pass to both callback functions.

mode (IN)
Specify `OCI_DEFAULT` fore now.

Comments

Return code: `OCI_ERROR` or `OCI_SUCCESS`.

The `createlcr_cb` argument must be of type `OCICallbackXStreamInLCRCreate`:

```
typedef sb4  (*OCICallbackXStreamInLCRCreate)
              void  *usrctxp, void **lcrp, ub1 *lcrtyp, oraub8 *flag);
```

Parameters of `OCICallbackXStreamInLCRCreate()`:

usrctxp (IN/OUT)
Pointer to the user context.

lcrp (OUT)
Pointer to the LCR to be sent.

lcrtyp (OUT)
LCR type (OCI_LCR_XROW or OCI_LCR_XDDL).

flag (OUT)
If OCI_XSTREAM_MORE_ROW_DATA is set, then the current LCR has more chunk data.

The input parameter to the callback is the user context. The output parameters are the new LCR, its type, and a flag. If the generated LCR contains additional chunk data, then this flag must have the OCI_XSTREAM_MORE_ROW_DATA (0x01) bit set. The valid return codes from the OCICallbackXStreamInLCRCreate() callback function are OCI_CONTINUE or OCI_SUCCESS. This callback function must return OCI_CONTINUE to continue processing the OCIXStreamInLCRCallbackSend() call. Any return code other than OCI_CONTINUE signals that the client wants to terminate the OCIXStreamInLCRCallbackSend() call immediately. In addition, a NULL LCR returned from the OCICallbackXStreamInLCRCreate() callback function signals that the client wants to terminate the current call.

The createchunk_cb argument must be of type OCICallbackXStreamInChunkCreate:

```
typedef sb4 (*OCICallbackXStreamInChunkCreate)
void     *usrctxp,
oratext  **column_name,
ub2      *column_name_len,
ub2      *column_dty,
oraub8   *column_flag,
ub2      *column_csid,
ub4      *chunk_bytes,
ub1      **chunk_data,
oraub8   *flag);
```

The input parameters of the createchunk_cb() procedure are the user context and the information about the chunk.

Parameters of OCICallbackXStreamInChunkCreate():

usrctxp (IN/OUT)
Pointer to the user context.

column_name (OUT)
Column name of the current chunk.

column_name_len (OUT)
Length of the column name.

column_name_dty (OUT)
Chunk data type (SQLT_CHR or SQLT_BIN).

column_flag (OUT)
See Comments in "OCIXStreamInChunkSend()" on page 11-55.

column_csid (OUT)
Column character set ID. Relevant only if the column is an XMLType column (that is, column_flag has the OCI_LCR_COLUMN_XML_DATA bit set).

chunk_bytes (OUT)
Chunk data length in bytes.

chunk_data (OUT)
Chunk data pointer.

flag (OUT)
If `OCI_XSTREAM_MORE_ROW_DATA` is set, then the current LCR has more chunk data.

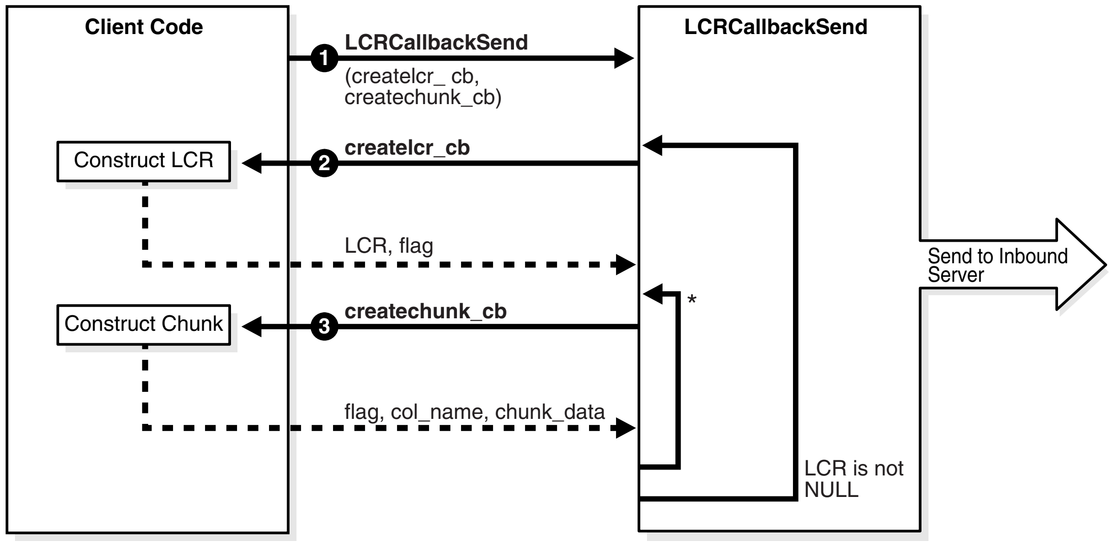
The `OCIXStreamInLCRCallbackSend()` function invokes the `createlcr_cb()` procedure to obtain each LCR to send to the server. If the return flag from the `createlcr_cb()` procedure has the `OCI_XSTREAM_MORE_ROW_DATA` bit set, then it invokes the `createchunk_cb()` procedure to obtain each chunk. It repeatedly calls the `createchunk_cb()` procedure while the flag returned from this callback has the `OCI_XSTREAM_MORE_ROW_DATA` bit set. When this bit is not set, this function cycles back to invoke the `createlcr_cb()` procedure to get the next LCR. This cycle is repeated until the `createlcr_cb()` procedure returns a `NULL` LCR or when at the transaction boundary after an ACK interval has elapsed since the call began.

The valid return codes from the `OCICallbackXStreamInChunkCreate()` callback function are `OCI_CONTINUE` or `OCI_SUCCESS`. This callback function must return `OCI_CONTINUE` to continue processing the `OCIXStreamInLCRCallbackSend()` call. Any return code other than `OCI_CONTINUE` signals that the client wants to terminate the `OCIXStreamInLCRCallbackSend()` call immediately.

Because terminating the current call flushes the network and incurs another network round-trip in the next call, you must avoid returning a `NULL` LCR immediately when there is no LCR to send. Doing this can greatly reduce network throughput and affect performance. During short idle periods, you can add some delays in the callback procedure instead of returning a `NULL` LCR immediately to avoid flushing the network too frequently.

Figure 11–1 shows the execution flow of the `OCIXStreamInLCRCallbackSend()` function.

Figure 11–1 Execution Flow of the OCIXStreamInLCRCallbackSend() Function



* `While OCI_XSTREAM_MORE_ROW_DATA` is set

Description of Figure 11–1:

- At 1, the user invokes the `OCIXStreamInLCRCallbackSend()` providing two callbacks. This function initiates an LCR inbound stream to the server.
- At 2, this function invokes the `createlcr_cb()` procedure to get an LCR from the callback to send to the server. If the return LCR is `NULL`, then this function exits.

- If the flag from 2 indicates the current LCR has more data (that is, the `OCI_XSTREAM_MORE_ROW_DATA` bit is set), then this function proceeds to 3; otherwise, it loops back to 2 to get the next LCR.
- At 3, this function invokes `createchunk_cb()` to get the chunk data to send to the server. If the flag from this callback has the `OCI_XSTREAM_MORE_ROW_DATA` bit set, then it repeats 3; otherwise, it loops back to 2 to get the next LCR from the user. If any callback function returns any values other than `OCI_CONTINUE`, then the `OCIXStreamInLCRCallbackSend()` call terminates.

  Following is a sample client pseudocode snippet for callback mode (error checking is not included for simplicity):

```
main
{
   /* Attach to inbound server */
   OCIXStreamInAttach();

   /* Get the server's processed low position to determine
    * the position of the first LCR to generate.
    */
   OCIXStreamInProcessedLWMGet(&lwm);

   while (TRUE)
   {
      /* Initiate LCR inbound stream */
      OCIXStreamInLCRCallbackSend(createlcr_cb, createchunk_cb);

      OCIXStreamInProcessedLWMGet(&lwm);

      if (some terminating condition)
         break;
   }
   OCIXStreamInDetach(&lwm);
}


createlcr_cb (IN usrctx, OUT lcr, OUT flag)
{
   if (have more LCRs to send)
   {
      /* construct lcr */
      OCILCRHeaderSet(lcr);
      OCILCRRowColumnInfoSet(lcr);

      if (lcr has LOB | LONG | XMLType columns)
         Set OCI_XSTREAM_MORE_ROW_DATA flag;

      if (lcr is LOB_ERASE | LOB_TRIM | LOB_WRITE)
         OCILCRLobInfoSet(lcr);
   }
   else if (idle timeout expires)
   {
      lcr = null;
   }
}

createchunk_cb (IN usrctx, OUT chunk, OUT flag)
{
   /* set col_name, col_flag, chunk data, and so on */
   construct_chunk;
```

```
            if (last chunk of current column)
            {
               set col_flag |= OCI_LCR_COLUMN_LAST_CHUNK;

               if (last column)
                  clear OCI_XSTREAM_MORE_ROW_DATA flag;
            }
         }
```

OCIXStreamInProcessedLWMGet()

Purpose

Gets the local processed low position that is cached at the client. This function can be called anytime while the client is attached to an XStream inbound server. Clients, using the callback mode to stream LCRs to the server (see "OCIXStreamInLCRCallbackSend()" on page 11-46), can invoke this function while in the callback procedures.

Syntax

```
sword OCIXStreamInProcessedLWMGet ( OCISvcCtx  *svchp,
                                    OCIError   *errhp,
                                    ub1        *processed_low_position,
                                    ub2        *processed_low_position_len,
                                    ub4        mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

processed_low_position (OUT)
The processed low position maintained at the client.

processed_low_position_len (OUT)
Length of `processed_low_position`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

After attaching to an XStream inbound server, a local copy of the server's processed low position (see "OCIXStreamOutProcessedLWMSet()" on page 11-70) is cached at the client. This local copy is refreshed with the server's low position when each of the following calls returns `OCI_SUCCESS`:

- `OCIXStreamInAttach()`
- `OCIXStreamInLCRSend()`
- `OCIXStreamInLCRCallbackSend()`
- `OCIXStreamInFlush()`

Return code: `OCI_ERROR` or `OCI_SUCCESS`.

You must pass in a preallocated buffer for the position argument. The maximum length of this buffer is `OCI_LCR_MAX_POSITION_LEN`. This position is exposed in the `DBA_XSTREAM_INBOUND_PROGRESS` view.

The client can use this position to periodically purge the logs used to generate the LCRs at or below this position.

OCIXStreamInErrorGet()

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

Purpose

Returns the first error encountered by the inbound server since the `OCIXStreamInAttach()` call.

Syntax

```
sword OCIXStreamInErrorGet ( OCISvcCtx *svchp,
                             OCIError  *errhp,
                             sb4       *errcodep,
                             oratext   *msgbuf,
                             ub2       msg_bufsize,
                             ub2       *msg_len,
                             oratext   *txn_id,
                             ub2       txn_id_bufsize,
                             ub2       *txn_id_len );
```

Parameters

svchp (IN/OUT)
OCI service handle.

errhp (IN/OUT)
Error Handle.

errcodep (OUT)
Error code.

msgbuf (IN/OUT)
Preallocated message buffer.

msg_bufsize (IN)
Message buffer size.

msg_len (OUT)
Length of returned error message.

txn_id (IN/OUT)
Preallocated transaction ID buffer.

txn_id_bufsize (IN)
The transaction ID buffer size.

txn_id_len (OUT)
Length of the returned transaction ID.

Comments

The maximum size for the returned transaction ID is `OCI_LCR_MAX_TXID_LEN`. If the allocated buffer for `txn_id` is too small, then this routine returns `ORA-29258`. The maximum size for the returned error msg is `OCI_ERROR_MAXMSG_SIZE`. If the allocated size for `msgbuf` is too small, then the returned message is truncated.

OCIXStreamInFlush()

Purpose

Used to flush the network while attaching to an XStream inbound server. It terminates any in-progress `OCIXStreamInLCRSend()` call associated with the specified service context.

Syntax

```
sword OCIXStreamInFlush ( OCISvcCtx    *svchp,
                          OCIError     *errhp,
                          ub4          mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

mode (IN)

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

`OCIXSTREAM_IN_FLUSH_WAIT_FOR_COMPLETE` - If this mode is specified, then this function flushes the network, and then waits for all complete and rollback transactions that have been sent to the inbound server to complete before returning control to the client.

Comments

Return code: `OCI_ERROR` or `OCI_SUCCESS`.

Each call incurs a database round-trip to get the server's processed low position, which you can retrieve afterward using `OCIXStreamInProcessedLWMGet()`. Call this function only when there is no LCR to send to the server and the client wants to know the progress of the attached inbound server.

This call returns `OCI_ERROR` if it is invoked from the callback functions of `OCIXStreamInLCRCallbackSend()`.

OCIXStreamInChunkSend()

Purpose

Sends a chunk to the inbound server. This function is valid during the execution of the `OCIXStreamInLCRSend()` call.

Syntax

```
sword OCIXStreamInChunkSend (  OCISvcCtx   *svchp,
                               OCIError    *errhp,
                               oratext     *column_name,
                               ub2         column_name_len,
                               ub2         column_dty,
                               oraub8      column_flag,
                               ub2         column_csid,
                               ub4         chunk_bytes,
                               ub1         *chunk_data,
                               oraub8      flag,
                               ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

column_name (IN)
Name of column associated with the given data. Column name must be canonicalized and must follow Oracle Database naming convention.

column_name_len (IN)
Length of column name.

column_dty (IN)
LCR chunk data type (must be `SQLT_CHR` or `SQLT_BIN`). See Table 11–5, " Storage of LOB or LONG Data in the LCR"

column_flag (IN)
Column flag. (See Comments for valid column flags.) Must specify `OCI_LCR_COLUMN_LAST_CHUNK` for the last chunk of each LOB or `LONG` or `XMLType` column.

column_csid (IN)
Column character set ID. This is required only if the `column_flag` has `OCI_LCR_COLUMN_XML_DATA` bit set.

chunk_bytes (IN)
Chunk data length in bytes.

chunk_data (IN)
Pointer to column data chunk. If the column is `NCLOB` or varying width `CLOB`, then the input chunk data must be in `AL16UTF16` format. The chunk data must be in the character set defined in " Storage of LOB or LONG Data in the LCR" on page 11-72.

flag (IN)
If OCI_XSTREAM_MORE_ROW_DATA (0x01) bit is set, then the current row change contains more data. You must clear this bit when sending the last chunk of the current LCR.

mode (IN)
Specify OCI_DEFAULT.

Comments

The following LCR column flags can be combined using bitwise OR operator.

```
#define OCI_LCR_COLUMN_LOB_DATA     /* column contains LOB data */
#define OCI_LCR_COLUMN_LONG_DATA    /* column contains long data */
#define OCI_LCR_COLUMN_EMPTY_LOB    /* column has an empty LOB  */
#define OCI_LCR_COLUMN_LAST_CHUNK   /* last chunk of current column */
#define OCI_LCR_COLUMN_AL16UTF16    /* column is in AL16UTF16 fmt */
#define OCI_LCR_COLUMN_NCLOB        /* column has NCLOB data */
#define OCI_LCR_COLUMN_XML_DATA     /* column contains xml data */
#define OCI_LCR_COLUMN_XML_DIFF     /* column contains xmldiff data */
#define OCI_LCR_COLUMN_ENCRYPTED    /* column is encrypted */
#define OCI_LCR_COLUMN_UPDATED      /* col is updated */
```

In Streams, LOB, LONG, or XMLType column data is broken up into multiple chunks. For a row change containing columns of these data types, its associated LCR only contains data for the other column types. All LOB, LONG or XMLType columns are either represented in the LCR as NULL or not included in the LCR as defined in Table 11–4, " Required Column List in the First LCR".

OCILCRRowColumnInfoSet() is provided to generate a list of scalar columns in an LCR. For LOB, LONG, and XMLType columns, OCIXStreamInChunkSend() is provided to set the value of each chunk in a column. For a large column, this function can be invoked consecutively multiple times with smaller chunks of data. The XStream inbound server can assemble these chunks and apply the accumulated change to the designated column.

The LCR of a row change must contain all the scalar columns that can uniquely identify a row at the apply site. Table 11–4 describes the required column list in each LCR for each DML operation.

Table 11–4 Required Column List in the First LCR

Command Type of the First LCR of a Row Change	Columns Required in the First LCR
INSERT	The NEW column list must contain all non-NULL scalar columns. All LOB, XMLType, and LONG columns with chunk data must be included in this NEW column list. Each must have NULL value and OCI_LCR_COLUMN_EMPTY_LOB flag specified.
UPDATE	The OLD column list must contain the key columns. The NEW column list must contain all updated scalar columns. All LOB, XMLType, and LONG columns with chunk data must be included in this NEW column list. Each must have NULL value and OCI_LCR_COLUMN_EMPTY_LOB flag specified.
DELETE	The OLD column list must contain the key columns.
LOB_WRITE, LOB_TRIM, LOB_ERASE	The NEW column list must contain the key columns and the modified LOB column.

After constructing each LCR, you can call `OCIXStreamInLCRSend()` to send that LCR. Afterward, `OCIXStreamInChunkSend()` can be called repeatedly to send the chunk data for each LOB or `LONG` or `XMLType` column in that LCR. Sending the chunk value for different columns cannot be interleaved. If a column contains multiple chunks, then this function must be called consecutively using the same column name before proceeding to a new column. The ordering of the columns is irrelevant.

When invoking this function, you must pass `OCI_XSTREAM_MORE_ROW_DATA` as the flag argument if there is more data for the current LCR. When sending the last chunk of the current LCR, then this flag must be cleared to signal the end of the current LCR.

This function is valid only for `INSERT`, `UPDATE`, and `LOB_WRITE` operations. Multiple LOB, `LONG`, or `XMLType` columns can be specified for `INSERT` and `UPDATE`, while only one LOB column is allowed for `LOB_WRITE` operation.

The following is a sample client pseudocode snippet for non-callback mode (error checking is not included for simplicity):

```
main
{
   /* Attach to inbound server */
   OCIXStreamInAttach();

   /* Get the server's processed low position to determine
    * the position of the first LCR to generate.
    */
   OCIXStreamInProcessedLWMGet(&lwm);

   while (TRUE)
   {
      flag = 0;
      /* construct lcr */
      OCILCRHeaderSet(lcr);
      OCILCRRowColumnInfoSet(lcr);

      if (lcr has LOB | LONG | XMLType columns)
         set OCI_XSTREAM_MORE_ROW_DATA flag;

      status = OCIXStreamInLCRSend(lcr, flag);

      if (status == OCI_STILL_EXECUTING &&
             (OCI_XSTREAM_MORE_ROW_DATA flag set))
      {
         for each LOB/LONG/XMLType column in row change
         {
            for each chunk in column
            {
               /* set col_name, col_flag, chunk data */
               construct chunk;

               if (last chunk of current column)
                   col_flag |= OCI_LCR_COLUMN_LAST_CHUNK;

               if (last chunk of last column)
                  clear OCI_XSTREAM_MORE_ROW_DATA flag;

               OCIXStreamInChunkSend(chunk, col_flag, flag);
            }
         }
      }
      else if (status == OCI_SUCCESS)
      {
         /* get lwm when SendLCR call ends successfully. */
         OCIXStreamInProcessedLWMGet(&lwm);
      }

      if (some terminating_condition)
        break;
   }

   OCIXStreamInDetach();
}
```

OCIXStreamInCommit()

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

Purpose

Commits the given transaction. This function lets the client notify the inbound server about a transaction that has been executed by the client rather than by the server. So that if the same transaction is retransmitted during apply restart, it is ignored by the inbound server. A commit LCR must be supplied for the inbound server to extract the transaction ID and the position of the commit.

Syntax

```
sword OCIXStreamInCommit ( OCISvcCtx *svchp,
                           OCIError  *errhp,
                           void      *lcrp,
                           ub4       mode );
```

Parameters

svchp (IN/OUT)
OCI service handle.

errhp (IN/OUT)
Error Handle to which errors should be reported.

lcrp (IN)
Pointer to the LCR to send. Must be a commit LCR.

mode (IN)
Mode flags. Not used currently; used for future extension.

Comments

The position of the input LCR must be higher than `DBA_XSTREAM_INBOUND_PROGRESS.APPLIED_HIGH_POSITION`, and the LCR's source database must match `DBA_APPLY_PROGRESS.SOURCE_DATABASE` of the attached inbound server.

If there is any pre-commit handler defined, it is executed when this commit LCR is executed.

Assume a sample use case in which a situation where the inbound server does not support certain data types, but the client can do the work directly. The client performs the transaction changes directly to the database and then invokes the `OCIXStreamInCommit()` to commit the transaction by way of the inbound server. Note that the client should not directly commit the transaction itself. Rather, the transaction changes are committed with this command (`OCIXStreamInCommit()`) so that the transaction is atomic. Thus, if the inbound server becomes disabled during the client transaction, then the entire transaction is correctly rolled back.

OCIXStreamOutAttach()

Purpose

Attaches to an XStream outbound server. The client application must connect to the database using a dedicated connection.

Syntax

```
sword OCIXStreamOutAttach ( OCISvcCtx   *svchp,
                            OCIError    *errhp,
                            oratext     *server_name,
                            ub2         server_name_len,
                            ub1         *last_position,
                            ub2         last_position_len,
                            ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

server_name (IN)
XStream outbound server name.

server_name_len (IN)
Length of XStream outbound server name.

last_position (IN)
Position to the last received LCR. Can be `NULL`.

last_position_len (IN)
Length of `last_position`.

mode (IN)

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

`OCIXSTREAM_OUT_ATTACH_APP_FREE_LCR` - If this mode is specified, then the application is in charge of freeing the LCRs from the outbound server.

Comments

The `OCIEnv` environment handle must be created with `OCI_OBJECT` mode, and the service context must be in a connected state to issue this function. This function does not support nonblocking mode. It returns either the `OCI_SUCCESS` or `OCI_ERROR` status code.

The name of the outbound server must be provided because multiple outbound servers can be configured in one Oracle Database instance. This function returns `OCI_ERROR` if it encounters any error while attaching to the outbound server. Only one client can attach to an XStream outbound server at any time. An error is returned if multiple clients attempt to attach to the same outbound server or if the same client attempts to attach to multiple outbound servers using the same service handle.

The `last_position` parameter is used to establish the starting point of the stream. This call returns `OCI_ERROR` if the specified position is non-`NULL` and less than the server's processed low position (see "OCIXStreamOutProcessedLWMSet()" on page 11-70); otherwise, LCRs with positions greater than the specified `last_position` are sent to the user.

If the `last_position` is `NULL`, then the stream starts from the processed low position maintained in the server.

OCIXStreamOutDetach()

Purpose

Detaches from the outbound server.

Syntax

```
sword OCIXStreamOutDetach ( OCISvcCtx   *svchp,
                            OCIError    *errhp,
                            ub4         mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle that you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

This function sends the current local processed low position to the server before detaching from the outbound server. The outbound server automatically restarts after this call. This function returns `OCI_ERROR` if it is invoked while a `OCIXStreamOutReceive()` call is in progress.

OCIXStreamOutLCRReceive()

Purpose

Receives an LCR from an outbound stream. If an LCR is available, then this function immediately returns that LCR. The duration of each LCR is limited to the interval between two successive `OCIXStreamOutLCRReceive()` calls. When there is no LCR available in the stream, this call returns a `NULL` LCR after an idle timeout.

Syntax

```
sword OCIXStreamOutLCRReceive ( OCISvcCtx     *svchp,
                                OCIError      *errhp,
                                void          **lcrp,
                                ub1           *lcrtype,
                                oraub8        *flag,
                                ub1           *fetch_low_position,
                                ub2           *fetch_low_position_len,
                                ub4           mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

lcrp (OUT)
Pointer to the LCR received from the stream. If there is an available LCR, then this LCR is returned with status code `OCI_STILL_EXECUTING`. When the call ends, a `NULL` LCR is returned with status code `OCI_SUCCESS`.

lcrtype (OUT)
Type of the retrieved LCR. This value is valid only when `lcrp` is not `NULL`.

flag (OUT)
Return flag. If bit `OCI_XSTREAM_MORE_ROW_DATA` (0x01) is set, then this LCR has more data. You must use `OCIXStreamOutReceiveChunk()` function to get the remaining data.

fetch_low_position (OUT)
XStream outbound server's fetch low position. This value is returned only when the return code is `OCI_SUCCESS`. Optional. If non-`NULL`, then you must preallocate `OCI_LCR_MAX_POSITION_LEN` bytes for the return value.

fetch_low_position_len (OUT)
Length of `fetch_low_position`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

To avoid a network round-trip for every `OCIXStreamOutLCRReceive()` call, the connection is tied to this call and allows the server to fill up the network buffer with LCRs so subsequent calls can quickly receive the LCRs from the network. The server ends each call at the transaction boundary after an ACK interval elapses since the call began. When there is no LCR in the stream, the server ends the call after the idle timeout elapses.

Return codes:

- `OCI_STILL_EXECUTING` means that the current call is still in progress. The connection associated with the specified service context handle is still tied to this call for streaming the LCRs from the server. An error is returned if you attempt to use the same connection to execute any OCI calls that require database round-trip, for example, `OCIStmtExecute()`, `OCIStmtFetch()`, `OCILobRead()`, and so on. `OCILCR*` calls do not require round-trips; thus, they are valid while the call is in progress.
- `OCI_SUCCESS` means that the current call is completed. You are free to execute `OCIStmt*`, `OCILob*`, and so on from the same service context.
- `OCI_ERROR` means the current call encounters some errors. Use `OCIErrorGet()` to obtain information about the error.

This call always returns a `NULL` LCR when the return code is `OCI_SUCCESS`. In addition, it returns the fetch low position to denote that the outbound server has received all transactions with commit position lower than or equal to this value.

See Also:

- "Server Handle Attributes" on page 10-3
- "OCIXStreamOutChunkReceive()" on page 11-71 for non-callback pseudocode in the Comments section

OCIXStreamOutLCRCallbackReceive()

Purpose

Used to get the LCR stream from the outbound server using callbacks. You must supply a callback procedure to be invoked for each LCR received. If some LCRs in the stream may contain LOB or LONG or XMLType columns, then a second callback must be supplied to process each chunk (see "OCIXStreamOutChunkReceive()" on page 11-71).

Syntax

```
sword OCIXStreamOutLCRCallbackReceive (
          OCISvcCtx                          *svchp,
          OCIError                           *errhp,
          OCICallbackXStreamOutLCRProcess    processlcr_cb,
          OCICallbackXStreamOutChunkProcess  processchunk_cb,
          void                               *usrctxp,
          ub1                                *fetch_low_position,
          ub2                                *fetch_low_position_len,
          ub4                                mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

processlcr_cb (IN)
Callback function to process each LCR received by the client. Cannot be `NULL`.

processchunk_cb (IN)
Callback function to process each chunk in the received LCR. Can be `NULL` if you do not expect to receive any LCRs with additional chunk data.

usrctxp (IN)
User context to pass to both callback procedures.

fetch_low_position (OUT)
XStream outbound server's fetch low position (see "OCIXStreamOutLCRReceive()" on page 11-63). Optional.

fetch_low_position_len (OUT)
Length of `fetch_low_position`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

Return code: `OCI_SUCCESS` or `OCI_ERROR`.

The `processlcr_cb` argument must be of type `OCICallbackXStreamOutLCRProcess`:

```
typedef sb4  (*OCICallbackXStreamOutLCRProcess)
               (void  *usrctxp, void *lcrp, ub1 lcrtyp, oraub8 flag);
```

Parameters of `OCICallbackXStreamOutLCRProcess()`:

usrctxp (IN/OUT)
Pointer to the user context.

lcrp (IN)
Pointer to the LCR just received.

lcrtyp (IN)
LCR type (`OCI_LCR_XROW` or `OCI_LCR_XDDL`).

flag (IN)
If `OCI_XSTREAM_MORE_ROW_DATA` is set, then the current LCR has more chunk data.

The input parameters of the `processlcr_cb()` procedure are the user context, the LCR just received, its type, and a flag to indicate whether the LCR contains more data. If there is an LCR available, then this callback is invoked immediately. If there is no LCR in the stream, after an idle timeout, then this call ends with `OCI_SUCCESS` return code. The valid return codes from the `OCICallbackXStreamOutLCRProcess()` callback function are `OCI_CONTINUE` or `OCI_SUCCESS`. This callback function must return `OCI_CONTINUE` to continue processing the `OCIXStreamOutLCRCallbackReceive()` call. Any return code other than `OCI_CONTINUE` signals that the client wants to terminate `OCIXStreamOutLCRCallbackReceive()` immediately.

See Also: "Server Handle Attributes" on page 10-3

The `processchunk_cb` argument must be of type `OCICallbackXStreamOutChunkProcess`:

```
typedef sb4  (*OCICallbackXStreamOutChunkProcess)
(void        *usrctxp,
oratext      *column_name,
ub2          column_name_len,
ub2          column_dty,
oraub8       column_flag,
ub2          column_csid,
ub4          chunk_bytes,
ub1          *chunk_data,
oraub8       flag );
```

Parameters of `OCICallbackXStreamOutChunkProcess()`:

usrctxp (IN/OUT)
Pointer to the user context.

column_name (IN)
Column name of the current chunk.

column_name_len (IN)
Length of the column name.

column_name_dty (IN)
Chunk data type (`SQLT_CHR` or `SQLT_BIN`).

column_flag (IN)
See Comments in "OCIXStreamInChunkSend()" on page 11-55.

column_csid (IN)
Column character set ID. Relevant only if the column is an `XMLType` column (that is, `column_flag` has the `OCI_LCR_COLUMN_XML_DATA` bit set).

chunk_bytes (IN)
Chunk data length in bytes.

chunk_data (IN)
Chunk data pointer.

flag (IN)
If `OCI_XSTREAM_MORE_ROW_DATA` is set, then the current LCR has more chunk data.

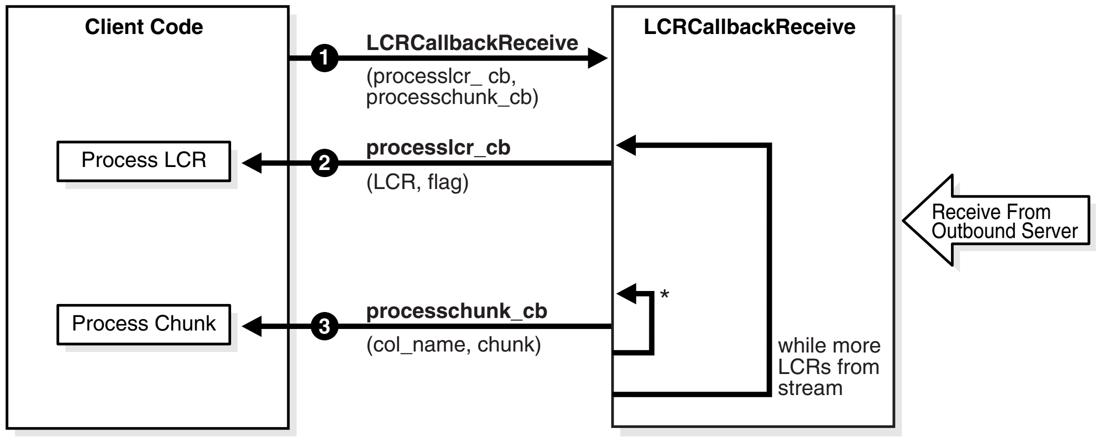
The input parameters of the `processchunk_cb()` procedure are the user context, the information about the chunk, and a flag. When the `flag` argument has the `OCI_XSTREAM_MORE_ROW_DATA` (0x01) bit set, then there is more data for the current LCR. The valid return codes from the `OCICallbackXStreamOutChunkProcess()` callback function are `OCI_CONTINUE` or `OCI_SUCCESS`. This callback function must return `OCI_CONTINUE` to continue processing the `OCIXStreamOutLCRCallbackReceive()` call. Any return code other than `OCI_CONTINUE` signals that the client wants to terminate `OCIXStreamOutLCRCallbackReceive()` immediately.

OCI calls are provided to access each field in the LCR. If the LCR contains only scalar column(s), then the duration of that LCR is limited only to the `processlcr_cb()` procedure. If the LCR contains some chunk data, then the duration of the LCR is extended until all the chunks have been processed. If you want to access the LCR data at a later time, then a copy of the LCR must be made before it is freed.

As for `OCIXStreamOutLCRReceive()`, the server ends each call at the transaction boundary after each ACK interval since the call began, or after each idle timeout. The default ACK interval is 30 seconds, and the default idle timeout is one second. See "Server Handle Attributes" on page 10-3 to tune these values. This function also returns the fetch low position when the call ends.

Figure 11–2 shows the execution flow of the `OCIXStreamOutLCRCallbackReceive()` function.

Figure 11–2 Execution Flow of the OCIXStreamOutLCRCallbackReceive() Function



* While `OCI_XSTREAM_MORE_ROW_DATA` is set.

Description of Figure 11–2:

- At 1, the client invokes `OCIXStreamOutLCRCallbackReceive()` providing two callbacks. This function initiates an LCR outbound stream from the server.
- At 2, this function receives an LCR from the stream and invokes `processlcr_cb()` procedure with the LCR just received. It passes `OCI_XSTREAM_MORE_ROW_DATA` flag to `processlcr_cb()` if the current LCR has additional data.
- If the current LCR has no additional chunk, then this function repeats 2 for the next LCR in the stream.
- At 3, if the current LCR contains additional chunk data, then this function invokes `processchunk_cb()` for each chunk received with the `OCI_XSTREAM_MORE_ROW_DATA` flag. This flag is cleared when the callback is invoked on the last chunk of the current LCR.
- If there is more LCR in the stream, then it loops back to 2. This process continues until the end of the current call, or when there is no LCR in the stream for one second, or if a callback function returns any value other than `OCI_CONTINUE`.

Here is sample pseudocode for callback mode:

```
main
{
   /* Attach to outbound server specifying last position */
   OCIXStreamOutAttach(last_pos);

   /* Update the local processed low position */
   OCIXStreamOutProcessedLWMSet(lwm);

   while (TRUE)
   {
      OCIXStreamOutLCRCallbackReceive(processlcr_cb,
                                      processchunk_cb, fwm);

      /* Use fetch low position(fwm)
       * to update processed lwm if applied.
       */

      /* Update the local lwm so it is sent to
       * server during next call.
       */
      OCIXStreamOutProcessedLWMSet(lwm);
      if (some terminating_condition)
         break;
   }
   OCIXStreamOutDetach();
}

processlcr_cb (IN lcr, IN flag)
{
   /* Process the LCR just received */
   OCILCRHeaderGet(lcr);
   OCILCRRowColumnInfoGet(lcr);

   if (lcr is LOB_WRITE | LOB_TRIM | LOB_ERASE)
      OCILCRLobInfoGet(lcr);

   if (OCI_XSTREAM_MORE_ROW_DATA flag set)
      prepare_for_chunk_data;
   else
      process_end_of_row;
}

processchunk_cb (IN chunk, IN flag)
{
   process_chunk;

   if (OCI_XSTREAM_MORE_ROW_DATA flag not set)
      process_end_of_row;
}
```

OCIXStreamOutProcessedLWMSet()

Purpose

Updates the local copy of the processed low position. This function can be called anytime between `OCIXStreamOutAttach()` and `OCIXStreamOutDetach()` calls. Clients using the callback mechanism to stream LCRs from the server (see "OCIXStreamOutLCRCallbackReceive()" on page 11-65), can invoke this function while in the callback procedures.

Syntax

```
sword OCIXStreamOutProcessedLWMSet ( OCISvcCtx  *svchp,
                                     OCIError   *errhp,
                                     ub1        *processed_low_position,
                                     ub2        processed_low_position_len,
                                     ub4        mode );
```

Parameters

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

processed_low_position (IN)
The processed low position maintained at the client.

processed_low_position_len (IN)
Length of `processed_low_position`.

mode (IN)
Specify `OCI_DEFAULT`.

Comments

The processed low position denotes that all LCRs at or below it have been processed. After successfully attaching to an XStream outbound server, a local copy of the processed low position is maintained at the client. Periodically, this position is sent to the server so that archived redo log files containing already processed transactions can be purged.

Return code: `OCI_SUCCESS` or `OCI_ERROR`.

Clients using `XStreamOut` functions must keep track of the processed low position based on what they have processed and call this function whenever their processed low position has changed. This is done so that a more current value is sent to the server during the next update, which occurs at the beginning of the `OCIXStreamOutLCRCallbackReceive()` and `OCIXStreamDetach()` calls. For an `OCIXStreamOutLCRReceive()` call, the processed low position is sent to the server when it initiates a request to start the outbound stream. It is not sent while the stream is in progress.

You can query the `DBA_XSTREAM_OUTBOUND_PROGRESS` view to confirm that the processed low position has been saved in the server.

OCIXStreamOutChunkReceive()

Purpose

Allows the client to retrieve the data of each LOB or `LONG` or `XMLType` column one chunk at a time.

Syntax

```
sword OCIXStreamOutChunkReceive ( OCISvcCtx   *svchp,
                                  OCIError    *errhp,
                                  oratext     **column_name,
                                  ub2         *column_name_len,
                                  ub2         *column_dty,
                                  oraub8      *column_flag,
                                  ub2         *column_csid,
                                  ub4         *chunk_bytes,
                                  ub1         **chunk_data,
                                  oraub8      *flag,
                                  ub4         mode );
```

Syntax

svchp (IN)
Service handle context.

errhp (IN/OUT)
An error handle you can pass to `OCIErrorGet()` for diagnostic information in case of an error.

column_name (OUT)
Name of the column that has data.

column_name_len (OUT)
Length of the column name.

column_dty (OUT)
Column chunk data type (either `SQLT_CHR` or `SQLT_BIN`).

column_flag (OUT)
Column flag. See Comments for valid flags.

column_csid (OUT)
Column character set ID. This is returned only for `XMLType` column, that is, `column_flag` has `OCI_LCR_COLUMN_XML_DATA` bit set.

chunk_bytes (OUT)
Number of bytes in the returned chunk.

chunk_data (OUT)
Pointer to the chunk data in the LCR. The client must not deallocate this buffer since the LCR and its contents are maintained by this function.

flag (OUT)
If `OCI_XSTREAM_MORE_ROW_DATA` (0x01) is set, then the current LCR has more chunks coming.

mode (IN)
Specify OCI_DEFAULT.

Comments

In Streams, LOB, LONG, or XMLType column data is broken up into multiple LCRs based on how they are stored in the online redo log files. Thus, for a row change containing these columns multiple LCRs may be constructed. The first LCR of a row change contains the column data for all the scalar columns. All LOB or LONG or XMLType columns in the first LCR are set to NULL because their data are sent in subsequent LCRs for that row change. These column data are stored in the LCR as either RAW (SQLT_BIN) or VARCHAR2 (SQLT_CHR) chunks as shown in the table Table 11–5.

Table 11–5 Storage of LOB or LONG Data in the LCR

Source Column Data Type	Streams LCR Data Type	Streams LCR Character Set
BLOB	RAW	N/A
Fixed-width CLOB	VARCHAR2	Client Character Set
Varying-width CLOB	RAW	AL16UTF16
NCLOB	RAW	AL16UTF16
XMLType	RAW	column csid obtained from the chunk

In Streams, LOB, LONG, or XMLType column data is broken up into multiple chunks based on how they are stored in the online redo log files. For a row change containing columns of these data types, its associated LCR only contains data for the other scalar columns. All LOB, LONG, or XMLType columns are either represented in the LCR as NULL or not included in the LCR. The actual data for these columns are sent following each LCR as RAW (SQLT_BIN) or VARCHAR2 (SQLT_CHR) chunks as shown in Table 11–5, " Storage of LOB or LONG Data in the LCR".

The following LCR column flags can be combined using the bitwise OR operator.

```
#define OCI_LCR_COLUMN_LOB_DATA     /* column contains LOB data */
#define OCI_LCR_COLUMN_LONG_DATA    /* column contains long data */
#define OCI_LCR_COLUMN_EMPTY_LOB    /* column has an empty LOB  */
#define OCI_LCR_COLUMN_LAST_CHUNK   /* last chunk of current column */
#define OCI_LCR_COLUMN_AL16UTF16    /* column is in AL16UTF16 fmt */
#define OCI_LCR_COLUMN_NCLOB        /* column has NCLOB data */
#define OCI_LCR_COLUMN_XML_DATA     /* column contains xml data */
#define OCI_LCR_COLUMN_XML_DIFF     /* column contains xmldiff data */
#define OCI_LCR_COLUMN_ENCRYPTED    /* column is encrypted */
#define OCI_LCR_COLUMN_UPDATED      /* col is updated */
```

Return code: OCI_ERROR or OCI_SUCCESS.

This call returns a NULL column name and NULL chunk data if it is invoked when the current LCR does not contain the LOB, LONG, or XMLType columns. This function is valid only when an OCIXStreamOutLCRReceive() call is in progress. An error is returned if it is called during other times.

If the return flag from OCIXStreamOutLCRReceive() has OCI_XSTREAM_MORE_ROW_DATA bit set, then you must iteratively call OCIXStreamOutChunkReceive() to retrieve all the chunks belonging to that row change before getting the next row change (that is, before making the next OCIXStreamOutLCRReceive() call); otherwise, an error is returned.

Here is sample pseudocode for non-callback mode:

```
main
{
   /* Attach to outbound server specifying last position */
   OCIXStreamOutAttach(last_pos);

   /* Update the local processed low position */
   OCIXStreamOutProcessedLWMSet(lwm);

   while (TRUE)
   {
      status = OCIXStreamOutLCRReceive(lcr, flag, fwm);

      if (status == OCI_STILL_EXECUTING)
      {
         /* Process LCR just received */
         OCILCRHeaderGet(lcr);
         OCILCRRowColumnInfoGet(lcr);

         while (OCI_XSTREAM_MORE_ROW_DATA flag set)
         {
            OCIXStreamReceiveChunk(chunk, flag, );

            process_chunk;
         }
         process_end_of_row;
      }
      else if (status == OCI_SUCCESS)
      {
         /* Use fetch low position(fwm)
          * to update processed lwm if applied.
          */

         /* Update the local lwm so it is sent to
          * server during next call.
          */
         OCIXStreamOutProcessedLWMSet(lwm);

          if (some terminating_condition)
            break;
       }
   }
   OCIXStreamOutDetach();
}
```

Part V

XStream Data Dictionary Views

This part contains descriptions of the data dictionary views related to XStream. This part contains the following chapters:

12 XStream Static Data Dictionary Views

This chapter describes the static data dictionary views related to XStream.

This chapter contains these topics:

See Also: *Oracle Database Reference*

ALL_APPLY

`ALL_APPLY` displays information about the apply processes that dequeue messages from queues accessible to the current user.

Related View

DBA_APPLY displays information about all apply processes in the database.

Column	Data Type	NULL	Description
APPLY_NAME	VARCHAR2(30)	NOT NULL	Name of the apply process
QUEUE_NAME	VARCHAR2(30)	NOT NULL	Name of the queue from which the apply process dequeues
QUEUE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the queue from which the apply process dequeues
APPLY_CAPTURED	VARCHAR2(3)		Indicates whether the apply process applies captured messages (YES) or user-enqueued messages (NO)
RULE_SET_NAME	VARCHAR2(30)		Name of the positive rule set used by the apply process for filtering
RULE_SET_OWNER	VARCHAR2(30)		Owner of the positive rule set used by the apply process for filtering
APPLY_USER	VARCHAR2(30)		User who is applying messages
APPLY_DATABASE_LINK	VARCHAR2(128)		Database link to which changes are applied. If NULL, then changes are applied to the local database.
APPLY_TAG	RAW(2000)		Tag associated with redo log records that are generated when changes are made by the apply process
DDL_HANDLER	VARCHAR2(98)		Name of the user-specified data definition language (DDL) handler, which handles DDL logical change records (LCRs)
PRECOMMIT_HANDLER	VARCHAR2(98)		Name of the user-specified pre-commit handler
MESSAGE_HANDLER	VARCHAR2(98)		Name of the user-specified procedure that handles dequeued messages other than LCRs
STATUS	VARCHAR2(8)		Status of the apply process: ■ DISABLED ■ ENABLED ■ ABORTED
MAX_APPLIED_MESSAGE_ NUMBER	NUMBER		System change number (SCN) corresponding to the apply process high watermark for the last time the apply process was stopped using the DBMS_APPLY_ADM.STOP_ APPLY procedure with the force parameter set to false. The apply process high watermark is the SCN beyond which no messages have been applied.
NEGATIVE_RULE_SET_NAME	VARCHAR2(30)		Name of the negative rule set used by the apply process for filtering
NEGATIVE_RULE_SET_OWNER	VARCHAR2(30)		Owner of the negative rule set used by the apply process for filtering
STATUS_CHANGE_TIME	DATE		Time that the STATUS of the apply process was changed
ERROR_NUMBER	NUMBER		Error number if the apply process was aborted
ERROR_MESSAGE	VARCHAR2(4000)		Error message if the apply process was aborted
MESSAGE_DELIVERY_MODE	VARCHAR2(10)		Reserved for internal use

Column	Data Type	NULL	Description
PURPOSE	VARCHAR2(19)		Purpose of the apply process: ■ Streams - An apply process in an Oracle Streams configuration ■ XStream Streams - An apply process in an Oracle Streams configuration with XStream capabilities enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure ■ XStream Out - An XStream outbound server in an XStream Out configuration ■ XStream In - An XStream inbound server in an XStream In configuration ■ AUDIT VAULT - An apply process in an audit vault configuration ■ CHANGE DATA CAPTURE - An apply process in a change data capture configuration

See Also: "DBA_APPLY" on page 12-11

ALL_APPLY_ERROR

ALL_APPLY_ERROR displays information about the error transactions generated by the apply processes that dequeue messages from queues accessible to the current user.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the ERROR_TYPE column is included in this view.

Related View

DBA_APPLY_ERROR displays information about the error transactions generated by all apply processes in the database.

Column	Data Type	NULL	Description
APPLY_NAME	VARCHAR2(30)		Name of the apply process at the local database which processed the transaction
QUEUE_NAME	VARCHAR2(30)		Name of the queue at the local database from which the transaction was dequeued
QUEUE_OWNER	VARCHAR2(30)		Owner of the queue at the local database from which the transaction was dequeued
LOCAL_TRANSACTION_ID	VARCHAR2(22)		Local transaction ID for the error transaction
SOURCE_DATABASE	VARCHAR2(128)		Database where the transaction originated
SOURCE_TRANSACTION_ID	VARCHAR2(128)		Original transaction ID at the source database
SOURCE_COMMIT_SCN	NUMBER		Original commit SCN for the transaction at the source database
MESSAGE_NUMBER	NUMBER		Identifier for the message in the transaction that raised an error
ERROR_NUMBER	NUMBER		Error number of the error raised by the transaction
ERROR_MESSAGE	VARCHAR2(4000)		Error message of the error raised by the transaction
RECIPIENT_ID	NUMBER		User ID of the original user that applied the transaction
RECIPIENT_NAME	VARCHAR2(30)		Name of the original user that applied the transaction
MESSAGE_COUNT	NUMBER		Total number of messages inside the error transaction
ERROR_CREATION_TIME	DATE		Time that the error was created

Column	Data Type	NULL	Description
SOURCE_COMMIT_POSITION	RAW(64)		Original commit position for the transaction
ERROR_TYPE	VARCHAR2(11)		NULL if the apply process can access all of the LCRs in the error transaction. When the ERROR_TYPE is NULL, manage the error transactions using the instructions in *Oracle Streams Concepts and Administration*. EAGER ERROR if the apply process cannot access all of the LCRs in the error transaction. This error type typically means that the apply process was applying LCRs in a large transaction. When the ERROR_TYPE is EAGER ERROR, manage the error transaction using the instructions in "Managing Eager Errors Encountered by an Inbound Server" on page 5-16.

See Also: "DBA_APPLY_ERROR" on page 12-11

ALL_APPLY_ERROR_MESSAGES

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

ALL_APPLY_ERROR_MESSAGES displays information about the individual messages in an error transaction generated by the apply processes that dequeue messages from queues accessible to the current user.

For XStream inbound servers, each message in an error transaction is an LCR.

Note:

- Messages that were spilled from memory to hard disk do not appear in this view.
- This view does not contain information related to XStream outbound servers.

RELATED VIEW

DBA_APPLY_ERROR_MESSAGES displays information about the individual messages in all of the error transactions generated by all apply processes in the database.

Column	Data Type	NULL	Description
MESSAGE_ID	RAW(16)		Unique identifier of the message stored in the error queue
LOCAL_TRANSACTION_ID	VARCHAR2(22)		Local transaction ID for the error transaction
TRANSACTION_MESSAGE_NUMBER	NUMBER		Message number of the message that raised the error The message number is a sequence number for the messages in the transaction, starting with 1.
ERROR_NUMBER	NUMBER		Error number of the error raised by the transaction The error number is populated only for the LCR that raised the error. This field is NULL for the other LCRs in the transaction.
ERROR_MESSAGE	VARCHAR2(4000)		Error message of the error raised by the transaction The error message is populated only for the LCR that raised the error. This field is NULL for the other LCRs in the transaction.

See Also: "DBA_APPLY_ERROR_MESSAGES" on page 12-11

ALL_CAPTURE

ALL_CAPTURE displays information about the capture processes that enqueue the captured changes into queues accessible to the current user.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the START_TIME and PURPOSE columns are included in this view.

RELATED VIEW

DBA_CAPTURE displays information about all capture processes in the database.

Column	Data Type	NULL	Description
CAPTURE_NAME	VARCHAR2(30)	NOT NULL	Name of the capture process
QUEUE_NAME	VARCHAR2(30)	NOT NULL	Name of the queue used for staging captured changes
QUEUE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the queue used for staging captured changes
RULE_SET_NAME	VARCHAR2(30)		Name of the positive rule set used by the capture process for filtering
RULE_SET_OWNER	VARCHAR2(30)		Owner of the positive rule set
CAPTURE_USER	VARCHAR2(30)		Current user who is enqueuing captured messages
START_SCN	NUMBER		SCN from which the capture process will start to capture changes
STATUS	VARCHAR2(8)		Status of the capture process: ■ DISABLED ■ ENABLED ■ ABORTED
CAPTURED_SCN	NUMBER		SCN of the last redo log record scanned
APPLIED_SCN	NUMBER		SCN of the most recent message dequeued by the relevant apply processes. All changes below this SCN have been dequeued by all apply processes that apply changes captured by this capture process.
USE_DATABASE_LINK	VARCHAR2(3)		Indicates whether the source database name is used as the database link to connect to the source database from the downstream database (YES) or not (NO). If the capture process was created at the source database, then this column will be NULL.
FIRST_SCN	NUMBER		SCN from which the capture process can be restarted
SOURCE_DATABASE	VARCHAR2(128)		Global name of the source database
SOURCE_DBID	NUMBER		Database ID of the source database
SOURCE_RESETLOGS_SCN	NUMBER		Resetlogs SCN of the source database
SOURCE_RESETLOGS_TIME	NUMBER		Resetlogs time of the source database
LOGMINER_ID	NUMBER		Session ID of the Oracle LogMiner session associated with the capture process
NEGATIVE_RULE_SET_NAME	VARCHAR2(30)		Name of the negative rule set used by the capture process for filtering
NEGATIVE_RULE_SET_OWNER	VARCHAR2(30)		Owner of the negative rule set used by the capture process for filtering
MAX_CHECKPOINT_SCN	NUMBER		SCN at which the last checkpoint was taken by the capture process

Column	Data Type	NULL	Description
REQUIRED_CHECKPOINT_SCN	NUMBER		Lowest SCN for which the capture process requires redo information to restart **Note:** This SCN value does not necessarily correspond with a checkpoint SCN value.
LOGFILE_ASSIGNMENT	VARCHAR2(8)		Logfile assignment type for the capture process: ■ IMPLICIT ■ EXPLICIT
STATUS_CHANGE_TIME	DATE		Time that the status of the capture process was changed
ERROR_NUMBER	NUMBER		Error number if the capture process was aborted
ERROR_MESSAGE	VARCHAR2(4000)		Error message if the capture process was aborted
VERSION	VARCHAR2(64)		Version number of the capture process
CAPTURE_TYPE	VARCHAR2(10)		Type of the capture process: ■ DOWNSTREAM ■ LOCAL
LAST_ENQUEUED_SCN	NUMBER		Last enqueued SCN
CHECKPOINT_RETENTION_TIME	NUMBER		Checkpoint retention time **Note:** When the checkpoint retention time for a capture process is set to INFINITE, then the value displayed in this column is 4294967295.
START_TIME	TIMESTAMP(6)		Time from which the capture process will start to capture changes
PURPOSE	VARCHAR2(19)		Purpose of the capture process: ■ Streams - A capture process in an Oracle Streams configuration ■ XStream Streams - A capture process in an Oracle Streams configuration with XStream capabilities enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure ■ XStream Out - A capture process in an XStream Out configuration ■ AUDIT VAULT - A capture process in an audit vault configuration ■ CHANGE DATA CAPTURE - A capture process in a change data capture configuration

See Also: "DBA_CAPTURE" on page 12-12

ALL_XSTREAM_INBOUND

ALL_XSTREAM_INBOUND displays information about the XStream inbound servers accessible to the current user.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the STATUS and COMMITTED_DATA_ONLY columns are included in this view.

Related View

DBA_XSTREAM_INBOUND displays information about all XStream inbound servers in the database.

Column	Data Type	NULL	Description
SERVER_NAME	VARCHAR2(30)	NOT NULL	Name of the inbound server
QUEUE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the queue associated with the inbound server
QUEUE_NAME	VARCHAR2(30)	NOT NULL	Name of the queue associated with the inbound server
APPLY_USER	VARCHAR2(30)		Name of the user who can connect to the inbound server and apply messages
USER_COMMENT	VARCHAR2(4000)		User comment
CREATE_DATE	TIMESTAMP(6)		Date when the inbound server was created
STATUS	VARCHAR2(8)		Status of the inbound server: ■ DISABLED - The inbound server is not running. ■ DETACHED - The inbound server is running, but the XStream client application is not attached to it. ■ ATTACHED - The inbound server is running, and the XStream client application is attached to it. ■ ABORTED - The inbound server became disabled because it encountered an error.
COMMITTED_DATA_ONLY	VARCHAR2(3)		YES if the inbound server can receive only LCRs in committed transactions from the XStream client application. A committed transaction is an assembled, noninterleaving transaction with no rollbacks. NO if the inbound server can receive LCRs in transactions that have not yet committed. This mode is for internal Oracle use only.

See Also: "DBA_XSTREAM_INBOUND" on page 12-12

ALL_XSTREAM_INBOUND_PROGRESS

ALL_XSTREAM_INBOUND_PROGRESS displays information about the progress made by the XStream inbound servers accessible to the current user.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the OLDEST_POSITION, OLDEST_MESSAGE_NUMBER, APPLIED_MESSAGE_NUMBER, APPLIED_TIME, APPLIED_MESSAGE_CREATE_TIME, SPILL_MESSAGE_NUMBER, and SOURCE_DATABASE columns are included in this view.

Related View

DBA_XSTREAM_INBOUND_PROGRESS displays information about the progress made by all XStream inbound servers in the database.

Column	Data Type	NULL	Description
SERVER_NAME	VARCHAR2(30)	NOT NULL	Name of the inbound server
PROCESSED_LOW_POSITION	RAW(64)		Position of the processed low transaction
APPLIED_LOW_POSITION	RAW(64)		All messages with a commit position less than this value have been applied
APPLIED_HIGH_POSITION	RAW(64)		Highest commit position of a transaction that has been applied
SPILL_POSITION	RAW(64)		Position of the spill low watermark of the transactions currently being applied
OLDEST_POSITION	RAW(64)		Earliest position of the transactions currently being applied

Column	Data Type	NULL	Description
OLDEST_MESSAGE_NUMBER	NUMBER	NOT NULL	Earliest message number of the transactions currently being applied
APPLIED_MESSAGE_NUMBER	NUMBER	NOT NULL	Message number up to which all transactions have definitely been applied. This value is the low watermark for the inbound server. That is, messages with a commit message number less than or equal to this message number have definitely been applied, but some messages with a higher commit message number may also have been applied.
APPLIED_TIME	DATE		Time at which the message with the message number displayed in the APPLIED_MESSAGE_NUMBER column was applied
APPLIED_MESSAGE_CREATE_TIME	DATE		Time at which the message with the message number displayed in the APPLIED_MESSAGE_NUMBER column was created at its source database
SPILL_MESSAGE_NUMBER	NUMBER		Spill low watermark. Any message with a lower SCN has either been applied or spilled to disk. The XStream client application does not need to send LCRs with a lower SCN than the spill low watermark. Spilled messages may not have been applied yet.
SOURCE_DATABASE	VARCHAR2(128)	NOT NULL	Database where the transaction originated

See Also: "DBA_XSTREAM_INBOUND_PROGRESS" on page 12-13

ALL_XSTREAM_OUTBOUND

ALL_XSTREAM_OUTBOUND displays information about the XStream outbound servers accessible to the current user.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the STATUS, COMMITTED_DATA_ONLY, START_SCN, and START_TIME columns are included in this view.

Related View

DBA_XSTREAM_OUTBOUND displays information about all XStream outbound servers in the database.

Column	Data Type	NULL	Description
SERVER_NAME	VARCHAR2(30)	NOT NULL	Name of the outbound server
CONNECT_USER	VARCHAR2(30)		Name of the user who can connect to the outbound server and process the outbound LCRs
CAPTURE_NAME	VARCHAR2(30)		Name of the Streams capture process
SOURCE_DATABASE	VARCHAR2(128)		Database where the transaction originated
CAPTURE_USER	VARCHAR2(30)		Current user who is enqueuing captured messages
QUEUE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the queue associated with the outbound server
QUEUE_NAME	VARCHAR2(30)	NOT NULL	Name of the queue associated with the outbound server
USER_COMMENT	VARCHAR2(4000)		User comment
CREATE_DATE	TIMESTAMP(6)		Date when the outbound server was created

Column	Data Type	NULL	Description
STATUS	VARCHAR2(8)		Status of the outbound server: ■ DISABLED - The outbound server is not running. ■ DETACHED - The outbound server is running, but the XStream client application is not attached to it. ■ ATTACHED - The outbound server is running, and the XStream client application is attached to it. ■ ABORTED - The outbound server became disabled because it encountered an error.
COMMITTED_DATA_ONLY	VARCHAR2(3)		YES if the outbound server can send only LCRs in committed transactions to the XStream client application. A committed transaction is an assembled, noninterleaving transaction with no rollbacks. NO if the outbound server can send LCRs in transactions that have not yet committed to the XStream client application. This mode is for internal Oracle use only.
START_SCN	NUMBER		The SCN from which the outbound server's capture process started capturing changes when it was last started
START_TIME	TIMESTAMP(6)		The time from which the outbound server's capture process started capturing changes when it was last started

See Also: "DBA_XSTREAM_OUTBOUND" on page 12-13

ALL_XSTREAM_OUTBOUND_PROGRESS

ALL_XSTREAM_OUTBOUND_PROGRESS displays information about the progress made by the XStream outbound servers accessible to the current user.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the OLDEST_POSITION column is included in this view.

Related View

DBA_XSTREAM_OUTBOUND_PROGRESS displays information about the progress made by all XStream outbound servers in the database.

Column	Data Type	NULL	Description
SERVER_NAME	VARCHAR2(30)	NOT NULL	Name of the outbound server
SOURCE_DATABASE	VARCHAR2(128)		Database where the transaction originated
PROCESSED_LOW_POSITION	RAW(64)		Position of the low watermark transaction processed by the outbound server
PROCESSED_LOW_TIME	DATE		Time when the processed low position was last updated by the outbound server
OLDEST_POSITION	RAW(64)		The position of the earliest LCR that is required by the XStream client application

See Also: "DBA_XSTREAM_OUTBOUND_PROGRESS" on page 12-14

ALL_XSTREAM_RULES

ALL_XSTREAM_RULES displays information about the XStream rules accessible to the current user.

Related View

DBA_XSTREAM_RULES displays information about all XStream server rules in the database.

Column	Data Type	NULL	Description
STREAMS_NAME	VARCHAR2(30)		Name of the Streams process
STREAMS_TYPE	VARCHAR2(12)		Type of the Streams process: ■ CAPTURE ■ APPLY
STREAMS_RULE_TYPE	VARCHAR2(6)		The Streams type of the rule: ■ TABLE ■ SCHEMA ■ GLOBAL
RULE_SET_OWNER	VARCHAR2(30)		Owner of the rule set
RULE_SET_NAME	VARCHAR2(30)		Name of the rule set
RULE_SET_TYPE	CHAR(8)		Type of the rule set: ■ POSITIVE ■ NEGATIVE
RULE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the rule
RULE_NAME	VARCHAR2(30)	NOT NULL	Name of the rule
RULE_TYPE	VARCHAR2(3)		The type of the rule: ■ DML ■ DDL
RULE_CONDITION	CLOB		Current rule condition
SCHEMA_NAME	VARCHAR2(30)		For table and schema rules, the schema name
OBJECT_NAME	VARCHAR2(30)		For table rules, the table name
INCLUDE_TAGGED_LCR	VARCHAR2(3)		Indicates whether to include tagged LCRs (YES) or not (NO)
SUBSETTING_OPERATION	VARCHAR2(6)		For subset rules, the type of operation: ■ INSERT ■ UPDATE ■ DELETE
DML_CONDITION	VARCHAR2(4000)		For subset rules, the row subsetting condition
SOURCE_DATABASE	VARCHAR2(128)		The name of the database where the LCRs originated
ORIGINAL_RULE_CONDITION	VARCHAR2(4000)		For rules created by Streams administrative APIs, the original rule condition when the rule was created
SAME_RULE_CONDITION	VARCHAR2(3)		For rules created by Streams administrative APIs, indicates whether the current rule condition is the same as the original rule condition (YES) or not (NO)

See Also: "DBA_XSTREAM_RULES" on page 12-14

DBA_APPLY

`DBA_APPLY` displays information about all apply processes in the database. Its columns are the same as those in `ALL_APPLY`.

See Also: "ALL_APPLY" on page 12-1

DBA_APPLY_ERROR

`DBA_APPLY_ERROR` displays information about the error transactions generated by all apply processes in the database. Its columns are the same as those in `ALL_APPLY_ERROR`.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `ERROR_TYPE` column is included in this view.

See Also: "ALL_APPLY_ERROR" on page 12-3

DBA_APPLY_ERROR_MESSAGES

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

`DBA_APPLY_ERROR_MESSAGES` displays information about the individual messages in all of the error transactions generated by all apply processes in the database. Its columns are the same as those in `ALL_APPLY_ERROR_MESSAGES`.

For XStream inbound servers, each message in an error transaction is an LCR.

Note:

- Messages that were spilled from memory to hard disk do not appear in this view.
- This view does not contain information related to XStream outbound servers.

See Also: "ALL_APPLY_ERROR_MESSAGES" on page 12-4

DBA_APPLY_SPILL_TXN

`DBA_APPLY_SPILL_TXN` displays information about the transactions spilled from memory to hard disk by all apply processes in the database.

Column	Data Type	NULL	Description
`APPLY_NAME`	`VARCHAR2(30)`	`NOT NULL`	Name of the apply process that spilled one or more transactions
`XIDUSN`	`NUMBER`	`NOT NULL`	Transaction ID undo segment number
`XIDSLT`	`NUMBER`	`NOT NULL`	Transaction ID slot number
`XIDSQN`	`NUMBER`	`NOT NULL`	Transaction ID sequence number
`FIRST_SCN`	`NUMBER`	`NOT NULL`	SCN of the first message in the transaction

Column	Data Type	NULL	Description
MESSAGE_COUNT	NUMBER		Number of messages spilled for the transaction
FIRST_MESSAGE_CREATE_TIME	DATE		Source creation time of the first message in the transaction
SPILL_CREATION_TIME	DATE		Time the first message was spilled
FIRST_POSITION	RAW(64)		Position of the first message in this transaction This column is populated only for an XStream inbound server.
TRANSACTION_ID	VARCHAR2(128)		Transaction ID of the spilled transaction

DBA_CAPTURE

DBA_CAPTURE displays information about all capture processes in the database. Its columns are the same as those in ALL_CAPTURE.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the START_TIME and PURPOSE columns are included in this view.

See Also: "ALL_CAPTURE" on page 12-5

DBA_XSTREAM_ADMINISTRATOR

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

DBA_XSTREAM_ADMINISTRATOR displays information about the users who have been granted privileges to be XStream administrators by procedures in the DBMS_XSTREAM_AUTH package.

Column	Data Type	NULL	Description
USERNAME	VARCHAR2(30)	NOT NULL	Name of the user who has been granted privileges to be an XStream administrator
LOCAL_PRIVILEGES	VARCHAR2(3)		Indicates whether the user has been granted local XStream administrator privileges (YES) or not (NO)
ACCESS_FROM_REMOTE	VARCHAR2(3)		Indicates whether the user can be used for remote XStream administration through a database link (YES) or not (NO)

See Also:

- Chapter 9, "DBMS_XSTREAM_AUTH"
- "Granting Privileges for the XStream Administrator" on page 4-1

DBA_XSTREAM_INBOUND

DBA_XSTREAM_INBOUND displays information about all XStream inbound servers in the database. Its columns are the same as those in ALL_XSTREAM_INBOUND.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `STATUS` and `COMMITTED_DATA_ONLY` columns are included in this view.

See Also: "ALL_XSTREAM_INBOUND" on page 12-6

DBA_XSTREAM_INBOUND_PROGRESS

`DBA_XSTREAM_INBOUND_PROGRESS` displays information about the progress made by all XStream inbound servers in the database. Its columns are the same as those in `ALL_XSTREAM_INBOUND_PROGRESS`.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `OLDEST_POSITION`, `OLDEST_MESSAGE_NUMBER`, `APPLIED_MESSAGE_NUMBER`, `APPLIED_TIME`, `APPLIED_MESSAGE_CREATE_TIME`, `SPILL_MESSAGE_NUMBER`, and `SOURCE_DATABASE` columns are included in this view.

See Also: "ALL_XSTREAM_INBOUND_PROGRESS" on page 12-7

DBA_XSTREAM_OUT_SUPPORT_MODE

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

`DBA_XSTREAM_OUT_SUPPORT_MODE` displays information about the level of capture process support for the tables in the database.

Column	Data Type	NULL	Description
`OWNER`	`VARCHAR2(30)`		Table owner
`OBJECT_NAME`	`VARCHAR2(30)`		Table name
`SUPPORT_MODE`	`VARCHAR2(6)`		Capture process support level for the table: ■ `FULL` - A capture process can capture changes made to all of the columns in the table. ■ `ID KEY` - A capture process can capture changes made to the key columns and any other columns in the table that are supported by the capture process, except for LOB, `LONG`, `LONG RAW`, and `XMLType` columns. ■ `NONE` - A capture process cannot capture changes made to any columns in the table.

DBA_XSTREAM_OUTBOUND

`DBA_XSTREAM_OUTBOUND` displays information about all XStream outbound servers in the database. Its columns are the same as those in `ALL_XSTREAM_OUTBOUND`.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `STATUS`, `COMMITTED_DATA_ONLY`, `START_SCN`, and `START_TIME` columns are included in this view.

See Also: "ALL_XSTREAM_OUTBOUND" on page 12-8

DBA_XSTREAM_OUTBOUND_PROGRESS

`DBA_XSTREAM_OUTBOUND_PROGRESS` displays information about the progress made by all XStream outbound servers in the database. Its columns are the same as those in `ALL_XSTREAM_OUTBOUND_PROGRESS`.

Note: Starting with Oracle Database 11*g* Release 2 (11.2.0.2), the `OLDEST_POSITION` column is included in this view.

See Also: "ALL_XSTREAM_OUTBOUND_PROGRESS" on page 12-14

DBA_XSTREAM_RULES

`DBA_XSTREAM_RULES` displays information about all XStream rules in the database. Its columns are the same as those in `ALL_XSTREAM_RULES`.

See Also: "ALL_XSTREAM_RULES" on page 12-10

13 XStream Dynamic Performance (V$) Views

This chapter describes the dynamic performance (V$) views related to XStream. In an XStream configuration, an apply process can function as an XStream outbound server or inbound server.

This chapter contains these topics:

See Also: *Oracle Database Reference*

V$STREAMS_APPLY_COORDINATOR

`V$STREAMS_APPLY_COORDINATOR` displays information about each apply process coordinator. The coordinator for an apply process gets transactions from the apply process reader and passes them to apply servers. An apply process coordinator is a subcomponent of an apply process, outbound server, or inbound server.

Column	Data Type	Description
`SID`	`NUMBER`	Session ID of the coordinator's session
`SERIAL#`	`NUMBER`	Serial number of the coordinator's session
`STATE`	`VARCHAR2(21)`	State of the coordinator: ■ `INITIALIZING` - Starting up ■ `IDLE` - Performing no work ■ `APPLYING` - Passing transactions to apply servers ■ `SHUTTING DOWN CLEANLY` - Stopping without an error ■ `ABORTING` - Stopping because of an apply error
`APPLY#`	`NUMBER`	Apply process number An apply process coordinator is an Oracle background process, prefixed by `ap`.
`APPLY_NAME`	`VARCHAR2(30)`	Name of the apply process
`TOTAL_APPLIED`	`NUMBER`	Total number of transactions applied by the apply process since the apply process was last started

Column	Data Type	Description
TOTAL_WAIT_DEPS	NUMBER	Number of times since the apply process was last started that an apply server waited to apply a logical change record (LCR) in a transaction until another apply server applied a transaction because of a dependency between the transactions
TOTAL_WAIT_COMMITS	NUMBER	Number of times since the apply process was last started that an apply server waited to commit a transaction until another apply server committed a transaction to serialize commits
TOTAL_ADMIN	NUMBER	Number of administrative jobs issued since the apply process was last started
TOTAL_ASSIGNED	NUMBER	Number of transactions assigned to apply servers since the apply process was last started
TOTAL_RECEIVED	NUMBER	Total number of transactions received by the coordinator process since the apply process was last started
TOTAL_IGNORED	NUMBER	Number of transactions which were received by the coordinator but were ignored because they had been previously applied
TOTAL_ROLLBACKS	NUMBER	Number of transactions which were rolled back due to unexpected contention
TOTAL_ERRORS	NUMBER	Number of transactions applied by the apply process that resulted in an apply error since the apply process was last started
UNASSIGNED_COMPLETE_TXNS	NUMBER	Total number of complete transactions that the coordinator has not assigned to any apply servers
AUTO_TXN_BUFFER_SIZE	NUMBER	Current value of transaction buffer size Transaction buffer size refers to the number of transactions that the apply reader can assemble ahead of apply servers. The apply process periodically adjusts the transaction buffer size.
LWM_TIME	DATE	Time when the message with the lowest message number was recorded The creation time of the message with the lowest message number was also recorded at this time.
LWM_MESSAGE_NUMBER	NUMBER	Number of the message corresponding to the low-watermark That is, messages with a commit message number less than or equal to this message number have definitely been applied, but some messages with a higher commit message number also may have been applied.
LWM_MESSAGE_CREATE_TIME	DATE	For captured messages, creation time at the source database of the message corresponding to the low-watermark. For user-enqueued messages, time when the message corresponding to the low-watermark was enqueued into the queue at the local database.
HWM_TIME	DATE	Time when the message with the highest message number was recorded The creation time of the message with the highest message number was also recorded at this time.
HWM_MESSAGE_NUMBER	NUMBER	Number of the message corresponding to the high-watermark That is, no messages with a commit message number greater than this message number have been applied.
HWM_MESSAGE_CREATE_TIME	DATE	For captured messages, creation time at the source database of the message corresponding to the high-watermark. For user-enqueued messages, time when the message corresponding to the high-watermark was enqueued into the queue at the local database.
STARTUP_TIME	DATE	Time when the apply process was last started
ELAPSED_SCHEDULE_TIME	NUMBER	Time elapsed (in hundredths of a second) scheduling messages since the apply process was last started
ELAPSED_IDLE_TIME	NUMBER	Elapsed idle time
LWM_POSITION	RAW(64)	Position of the low-watermark LCR
HWM_POSITION	RAW(64)	Position of the high-watermark LCR
PROCESSED_MESSAGE_NUMBER	NUMBER	Message number currently processed by the apply coordinator

Note: The ELAPSED_SCHEDULE_TIME column is only populated if the TIMED_STATISTICS initialization parameter is set to true, or if the STATISTICS_LEVEL initialization parameter is set to TYPICAL or ALL.

V$STREAMS_APPLY_READER

V$STREAMS_APPLY_READER displays information about each apply reader. The apply reader is a process which reads (dequeues) messages from the queue, computes message dependencies, and builds transactions. It passes the transactions on to the coordinator in commit order for assignment to the apply servers. An apply reader is a subcomponent of an apply process, outbound server, or inbound server.

Column	Data Type	Description
SID	NUMBER	Session ID of the reader's session
SERIAL#	NUMBER	Serial number of the reader's session
APPLY#	NUMBER	Apply process number An apply process is an Oracle background process prefixed by ap.
APPLY_NAME	VARCHAR2(30)	Name of the apply process
STATE	VARCHAR2(36)	State of the reader: ■ INITIALIZING - Starting up. ■ IDLE - Performing no work. ■ DEQUEUE MESSAGES - Dequeuing messages from the queue. ■ SCHEDULE MESSAGES - Computing dependencies between messages and assembling messages into transactions. ■ SPILLING - Spilling unapplied messages from memory to hard disk. ■ PAUSED - WAITING FOR DDL TO COMPLETE - Waiting for a data definition language (DDL) LCR to be applied.
TOTAL_MESSAGES_DEQUEUED	NUMBER	Total number of messages dequeued since the apply process was last started
TOTAL_MESSAGES_SPILLED	NUMBER	Number of messages spilled by the reader since the apply process was last started
DEQUEUE_TIME	DATE	Time when the last message was received
DEQUEUED_MESSAGE_NUMBER	NUMBER	Number of the last message received
DEQUEUED_MESSAGE_CREATE_TIME	DATE	For captured messages, creation time at the source database of the last message received. For user-enqueued messages, time when the message was enqueued into the queue at the local database.
SGA_USED	NUMBER	Amount (in bytes) of SGA memory used by the apply process since it was last started
ELAPSED_DEQUEUE_TIME	NUMBER	Time elapsed (in hundredths of a second) dequeuing messages since the apply process was last started
ELAPSED_SCHEDULE_TIME	NUMBER	Time elapsed (in hundredths of a second) scheduling messages since the apply process was last started. Scheduling includes computing dependencies between messages and assembling messages into transactions.
ELAPSED_SPILL_TIME	NUMBER	Elapsed time (in hundredths of a second) spent spilling messages since the apply process was last started
LAST_BROWSE_NUM	NUMBER	Reserved for internal use
OLDEST_SCN_NUM	NUMBER	Oldest SCN
LAST_BROWSE_SEQ	NUMBER	Reserved for internal use
LAST_DEQ_SEQ	NUMBER	Last dequeue sequence number

Column	Data Type	Description
OLDEST_XIDUSN	NUMBER	Transaction ID undo segment number of the oldest transaction that either has been applied or is being applied
OLDEST_XIDSLT	NUMBER	Transaction ID slot number of the oldest transaction that either has been applied or is being applied
OLDEST_XIDSQN	NUMBER	Transaction ID sequence number of the oldest transaction that either has been applied or is being applied
SPILL_LWM_SCN	NUMBER	Spill low-watermark SCN
PROXY_SID	NUMBER	When the apply process uses combined capture and apply, the session ID of the propagation receiver that is responsible for direct communication between capture and apply. If the apply process does not use combined capture and apply, then this column is `0`.
PROXY_SERIAL	NUMBER	When the apply process uses combined capture and apply, the serial number of the propagation receiver that is responsible for direct communication between capture and apply. If the apply process does not use combined capture and apply, then this column is `0`.
PROXY_SPID	VARCHAR2(12)	When the apply process uses combined capture and apply, the process identification number of the propagation receiver that is responsible for direct communication between capture and apply. If the apply process does not use combined capture and apply, then this column is `0`.
CAPTURE_BYTES_RECEIVED	NUMBER	When the apply process uses combined capture and apply, the number of bytes received by the apply process from the capture process since the apply process last started. If the apply process does not use combined capture and apply, then this column is not populated.
DEQUEUED_POSITION	RAW(64)	Dequeued position This column is populated only for an apply process that is functioning as an XStream inbound server.
LAST_BROWSE_POSITION	RAW(64)	Reserved for internal use
OLDEST_POSITION	RAW(64)	The earliest position of the transactions currently being dequeued and applied This column is populated only for an apply process that is functioning as an XStream inbound server.
SPILL_LWM_POSITION	RAW(64)	Spill low-watermark position This column is populated only for an apply process that is functioning as an XStream inbound server.
OLDEST_TRANSACTION_ID	VARCHAR2(128)	Oldest transaction ID
TOTAL_LCRS_WITH_DEP	NUMBER	Total number of LCRs with row-level dependencies since the apply process last started
TOTAL_LCRS_WITH_WMDEP	NUMBER	Total number of LCRs with watermark dependencies since the apply process last started A watermark dependency occurs when an apply process must wait until the apply process's low-watermark reaches a particular threshold.
TOTAL_IN_MEMORY_LCRS	NUMBER	Total number of LCRs currently in memory
SGA_ALLOCATED	NUMBER	The total amount of shared memory (in bytes) allocated from the Streams pool for the apply process since the apply process last started

Note: The `ELAPSED_DEQUEUE_TIME` and `ELAPSED_SCHEDULE_TIME` columns are only populated if the `TIMED_STATISTICS` initialization parameter is set to `true`, or if the `STATISTICS_LEVEL` initialization parameter is set to `TYPICAL` or `ALL`.

V$STREAMS_APPLY_SERVER

`V$STREAMS_APPLY_SERVER` displays information about each apply server and its activities. An apply server receives messages from the apply coordinator for an apply process. For each message received, an apply server either applies the message or sends the message to the appropriate apply handler. An apply server is a subcomponent of an apply process, outbound server, or inbound server.

Column	Data Type	Description
`SID`	`NUMBER`	Session ID of the apply server's session
`SERIAL#`	`NUMBER`	Serial number of the apply server's session
`APPLY#`	`NUMBER`	Apply process number An apply process is an Oracle background process prefixed by `ap`.
`APPLY_NAME`	`VARCHAR2(30)`	Name of the apply process
`SERVER_ID`	`NUMBER`	Parallel execution server number of the apply server
`STATE`	`VARCHAR2(20)`	State of the apply server: ■ `INITIALIZING` - Starting up. ■ `IDLE` - Performing no work. ■ `RECORD LOW-WATERMARK` - Performing an administrative job that maintains information about the apply progress, which is used in the `ALL_APPLY_PROGRESS` and `DBA_APPLY_PROGRESS` data dictionary views. ■ `ADD PARTITION` - Performing an administrative job that adds a partition that is used for recording information about in-progress transactions. ■ `DROP PARTITION` - Performing an administrative job that purges rows that were used to record information about in-progress transactions. ■ `EXECUTE TRANSACTION` - Applying a transaction. ■ `WAIT COMMIT` - Waiting to commit a transaction until all other transactions with a lower commit SCN are applied. This state is possible only if the `COMMIT_SERIALIZATION` apply process parameter is set to a value other than `DEPENDENT_TRANSACTIONS` and the `PARALLELISM` apply process parameter is set to a value greater than `1`. ■ `WAIT DEPENDENCY` - Waiting to apply an LCR in a transaction until another transaction, on which it has a dependency, is applied. This state is possible only if the `PARALLELISM` apply process parameter is set to a value greater than `1`. ■ `ROLLBACK TRANSACTION` - Rolling back a transaction. ■ `TRANSACTION CLEANUP` - Cleaning up an applied transaction, which includes removing LCRs from the apply process's queue. ■ `WAIT FOR CLIENT` - Waiting for an XStream client application to request more LCRs. ■ `WAIT FOR NEXT CHUNK` - Waiting for the next set of LCRs for a large transaction.
`XIDUSN`	`NUMBER`	Transaction ID undo segment number of the transaction currently being applied
`XIDSLT`	`NUMBER`	Transaction ID slot number of the transaction currently being applied
`XIDSQN`	`NUMBER`	Transaction ID sequence number of the transaction currently being applied
`COMMITSCN`	`NUMBER`	Commit SCN of the transaction currently being applied
`DEP_XIDUSN`	`NUMBER`	Transaction ID undo segment number of a transaction on which the transaction being applied by this apply server depends
`DEP_XIDSLT`	`NUMBER`	Transaction ID slot number of a transaction on which the transaction being applied by this apply server depends
`DEP_XIDSQN`	`NUMBER`	Transaction ID sequence number of a transaction on which the transaction being applied by this apply server depends

Column	Data Type	Description
DEP_COMMITSCN	NUMBER	Commit SCN of the transaction on which this apply server depends
MESSAGE_SEQUENCE	NUMBER	Number of the current message being applied by the apply server. This value is reset to 1 at the beginning of each transaction.
TOTAL_ASSIGNED	NUMBER	Total number of transactions assigned to the apply server since the apply process was last started
TOTAL_ADMIN	NUMBER	Total number of administrative jobs done by the apply server since the apply process was last started. See the STATE information in this view for the types of administrative jobs.
TOTAL_ROLLBACKS	NUMBER	Number of transactions assigned to this server that were rolled back
TOTAL_MESSAGES_APPLIED	NUMBER	Total number of messages applied by this apply server since the apply process was last started
APPLY_TIME	DATE	Time the last message was applied
APPLIED_MESSAGE_NUMBER	NUMBER	Number of the last message applied
APPLIED_MESSAGE_CREATE_TIME	DATE	Creation time at the source database of the last captured message applied. No information about user-enqueued messages is recorded in this column.
ELAPSED_DEQUEUE_TIME	NUMBER	Time elapsed (in hundredths of a second) dequeuing messages since the apply process was last started
ELAPSED_APPLY_TIME	NUMBER	Time elapsed (in hundredths of a second) applying messages since the apply process was last started
COMMIT_POSITION	RAW(64)	Commit position of the transaction. This column is populated only for an apply process that is functioning as an XStream outbound server or inbound server.
DEP_COMMIT_POSITION	RAW(64)	Commit position of the transaction the slave depends on This column is populated only for an apply process that is functioning as an XStream inbound server.
LAST_APPLY_POSITION	RAW(64)	For inbound servers, the position of the last message applied; for outbound servers, the position of the last message sent to the XStream client application This column is populated only for an apply process that is functioning as an XStream outbound server or inbound server.
TRANSACTION_ID	VARCHAR2(128)	Transaction ID that the slave is applying This column is populated only for an apply process that is functioning as an XStream inbound server.
DEP_TRANSACTION_ID	VARCHAR2(128)	Transaction ID of the transaction the slave depends on This column is populated only for an apply process that is functioning as an XStream inbound server.

Note:

- The `ELAPSED_DEQUEUE_TIME` and `ELAPSED_APPLY_TIME` columns are only populated if the `TIMED_STATISTICS` initialization parameter is set to `true`, or if the `STATISTICS_LEVEL` initialization parameter is set to `TYPICAL` or `ALL`.
- The `WAIT FOR NEXT CHUNK` apply server state is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

V$XSTREAM_CAPTURE

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

V$XSTREAM_CAPTURE displays information about each capture process that sends LCRs to an XStream outbound server.

Note: This view does not display information about capture processes that send LCRs to Oracle Streams apply processes. To view information about such capture processes, query the V$STREAMS_CAPTURE view.

Column	Data Type	Description
SID	NUMBER	Session identifier of the capture process
SERIAL#	NUMBER	Session serial number of the capture process session
CAPTURE#	NUMBER	Capture process number A capture process is an Oracle background process prefixed by cp.
CAPTURE_NAME	VARCHAR2(30)	Name of the capture process
LOGMINER_ID	NUMBER	Session ID of the Oracle LogMiner session associated with the capture process
STARTUP_TIME	DATE	Time when the capture process was last started

Column	Data Type	Description
STATE	VARCHAR2(551)	State of the capture process: ■ INITIALIZING - Starting up. ■ WAITING FOR DICTIONARY REDO - Waiting for redo log files containing the dictionary build related to the first SCN to be added to the capture process session. A capture process cannot begin to scan the redo log files until all of the log files containing the dictionary build have been added. ■ DICTIONARY INITIALIZATION - Processing a dictionary build. ■ MINING (PROCESSED SCN = *scn_value*) - Mining a dictionary build at the SCN *scn_value*. ■ LOADING (step *X* of *Y*) - Processing information from a dictionary build and currently at step *X* in a process that involves *Y* steps, where *X* and *Y* are numbers. ■ CAPTURING CHANGES - Scanning the redo log for changes that satisfy the capture process rule sets. ■ WAITING FOR REDO - Waiting for new redo log files to be added to the capture process session. The capture process has finished processing all of the redo log files added to its session. This state is possible if there is no activity at a source database. For a downstream capture process, this state is possible if the capture process is waiting for new log files to be added to its session. ■ EVALUATING RULE - Evaluating a change against a capture process rule set. ■ CREATING LCR - Converting a change into an LCR. ■ ENQUEUING MESSAGE - Enqueuing an LCR that satisfies the capture process rule sets into the capture process queue. ■ PAUSED FOR FLOW CONTROL - Unable to enqueue LCRs either because of low memory or because propagations and outbound servers are consuming messages slower than the capture process is creating them. This state indicates flow control that is used to reduce spilling of captured LCRs when propagation or apply has fallen behind or is unavailable. ■ WAITING FOR THE BUFFERED QUEUE TO SHRINK - Waiting for the buffered queue to change to a smaller size. The buffered queue shrinks when there is a memory limitation or when an administrator reduces its size. ■ WAITING FOR *n* SUBSCRIBER(S) INITIALIZING - Waiting for outbound servers that receive LCRs from the capture process to start, where *n* is the number of outbound servers. ■ WAITING FOR TRANSACTION - Waiting for LogMiner to provide more transactions. ■ WAITING FOR INACTIVE DEQUEUERS - Waiting for the capture process's queue subscribers to start. The capture process stops enqueuing LCRs if there are no active subscribers to the queue. ■ SUSPENDED FOR AUTO SPLIT/MERGE - Waiting for a merge operation to complete. ■ SHUTTING DOWN - Stopping. ■ ABORTING - Aborting.
TOTAL_PREFILTER_DISCARDED	NUMBER	Total number of prefiltered messages discarded
TOTAL_PREFILTER_KEPT	NUMBER	Total number of prefiltered messages kept
TOTAL_PREFILTER_EVALUATIONS	NUMBER	Total number of prefilter evaluations
TOTAL_MESSAGES_CAPTURED	NUMBER	Total number of redo entries passed by LogMiner to the capture process for detailed rule evaluation since the capture process last started. A capture process converts a redo entry into a message and performs detailed rule evaluation on the message when capture process prefiltering cannot discard the change.
CAPTURE_TIME	DATE	Time when the most recent message was captured
CAPTURE_MESSAGE_NUMBER	NUMBER	Number of the most recently captured message

Column	Data Type	Description
CAPTURE_MESSAGE_CREATE_TIME	DATE	Creation time of the most recently captured message
TOTAL_MESSAGES_CREATED	NUMBER	Count associated with ELAPSED_LCR_TIME to calculate rate
TOTAL_FULL_EVALUATIONS	NUMBER	Count associated with ELAPSED_RULE_TIME to calculate rate
TOTAL_MESSAGES_ENQUEUED	NUMBER	Total number of messages enqueued since the capture process was last started
ENQUEUE_TIME	DATE	Time when the last message was enqueued
ENQUEUE_MESSAGE_NUMBER	NUMBER	Number of the last enqueued message
ENQUEUE_MESSAGE_CREATE_TIME	DATE	Creation time of the last enqueued message
AVAILABLE_MESSAGE_NUMBER	NUMBER	For local capture, the last redo SCN flushed to the log files. For downstream capture, the last SCN added to LogMiner through the archived redo log files.
AVAILABLE_MESSAGE_CREATE_TIME	DATE	For local capture, the time the SCN was written to the log file. For downstream capture, the time the most recent archived redo log file (containing the most recent SCN) was added to LogMiner.
ELAPSED_CAPTURE_TIME	NUMBER	Elapsed time (in hundredths of a second) scanning for changes in the redo log since the capture process was last started
ELAPSED_RULE_TIME	NUMBER	Elapsed time (in hundredths of a second) evaluating rules since the capture process was last started
ELAPSED_ENQUEUE_TIME	NUMBER	Elapsed time (in hundredths of a second) enqueuing messages since the capture process was last started
ELAPSED_LCR_TIME	NUMBER	Elapsed time (in hundredths of a second) creating LCRs since the capture process was last started
ELAPSED_REDO_WAIT_TIME	NUMBER	Elapsed time (in hundredths of a second) spent by the capture process in the WAITING FOR REDO state
ELAPSED_PAUSE_TIME	NUMBER	Elapsed flow control pause time (in hundredths of a second)
STATE_CHANGED_TIME	DATE	Time at which the state of the capture process changed
SGA_USED	NUMBER	The total amount of shared memory (in bytes) currently used by the capture process out of the amount allocated (SGA_ALLOCATED)
SGA_ALLOCATED	NUMBER	The total amount of shared memory (in bytes) allocated from the Streams pool for the capture process
BYTES_OF_REDO_MINED	VARCHAR2(64)	The total amount of redo data mined (in bytes) since the capture process last started
SESSION_RESTART_SCN	VARCHAR2(64)	The SCN from which the capture process started mining redo data when it was last started

Note: The ELAPSED_CAPTURE_TIME, ELAPSED_RULE_TIME, ELAPSED_ENQUEUE_TIME, ELAPSED_LCR_TIME, and ELAPSED_REDO_WAIT_TIME columns are only populated if the TIMED_STATISTICS initialization parameter is set to true, or if the STATISTICS_LEVEL initialization parameter is set to TYPICAL or ALL.

V$XSTREAM_MESSAGE_TRACKING

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

V$XSTREAM_MESSAGE_TRACKING displays information about LCRs tracked through the stream that are processed by XStream components.

You can track an LCR through a stream using one of the following methods:

- Set the message_tracking_frequency capture process parameter to 1 or another relatively low value.
- Run the SET_MESSAGE_TRACKING procedure in the DBMS_STREAMS_ADM package.

  When the actions parameter in the DBMS_STREAMS_ADM.SET_MESSAGE_TRACKING procedure is set to DBMS_STREAMS_ADM.ACTION_MEMORY, information about the LCRs is tracked in memory, and this view is populated with information about the LCRs. Currently, DBMS_STREAMS_ADM.ACTION_MEMORY is the only valid setting for the actions parameter in the procedure.

Note: This view does not display information about messages flowing in an Oracle Streams configuration. To view information about such message streams, query the V$STREAMS_MESSAGE_TRACKING view.

Column	Data Type	Description
TRACKING_LABEL	VARCHAR2(30)	User-specified tracking label
TAG	RAW(30)	First 30 bytes of the tag of the LCR
COMPONENT_NAME	VARCHAR2(30)	Name of the component that processed the LCR
COMPONENT_TYPE	VARCHAR2(30)	Type of the component that processed the LCR
ACTION	VARCHAR2(50)	Action performed on the LCR
ACTION_DETAILS	VARCHAR2(100)	Details of the action
TIMESTAMP	TIMESTAMP(9) WITH TIME ZONE	Time when the action was performed
MESSAGE_CREATION_TIME	DATE	Time when the message was created
MESSAGE_NUMBER	NUMBER	SCN of the message
TRACKING_ID	RAW(16)	Globally unique OID of the LCR
SOURCE_DATABASE_NAME	VARCHAR2(128)	Name of the source database
OBJECT_OWNER	VARCHAR2(30)	Owner of the object
OBJECT_NAME	VARCHAR2(30)	Name of the object
XID	VARCHAR2(128)	Transaction ID
COMMAND_TYPE	VARCHAR2(30)	Command type of the LCR
MESSAGE_POSITION	RAW(64)	Position of the message

See Also: *Oracle Streams Replication Administrator's Guide*

V$XSTREAM_OUTBOUND_SERVER

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

V$XSTREAM_OUTBOUND_SERVER displays statistics about an outbound server. An outbound server sends LCRs to an XStream client application.

Note: When the COMMITTED_DATA_ONLY column is YES in the V$XSTREAM_OUTBOUND_SERVER view, the V$STREAMS_APPLY_SERVER view provides additional information about the outbound server process, and information about the apply server background processes used by the outbound server.

Column	Data Type	Description
SID	NUMBER	Session ID of the outbound server's session
SERIAL#	NUMBER	Serial number of the outbound server's session
SPID	VARCHAR2(12)	Process identification number of the operating-system process that sends LCRs to the client application
SERVER_NAME	VARCHAR2(30)	Name of the outbound server
STARTUP_TIME	DATE	Time when the client application attached to the outbound server

Column	Data Type	Description
STATE	VARCHAR2(37)	State of the outbound server When the COMMITTED_DATA_ONLY column shows YES, the following states are possible: ■ INITIALIZING - Starting up the outbound server. ■ IDLE - Performing no work because there are no LCRs to send to the XStream client application. ■ GET TRANSACTIONS - Receiving transactions from the outbound server's apply coordinator. ■ SEND TRANSACTION - Sending a transaction to an XStream client application. ■ WAIT FOR NEXT CHUNK - Waiting for the next set of LCRs for a large transaction. ■ TRANSACTION CLEANUP - Cleaning up an applied transaction, which includes removing LCRs from the outbound server's queue. ■ WAIT FOR CLIENT - Waiting for an XStream client application to request more LCRs. When the COMMITTED_DATA_ONLY column shows NO, the following states are possible: ■ INITIALIZING - Starting up the outbound server. ■ INITIALIZING RULE EVALUATION CONTEXT - Initializing the context to evaluate the outbound server's rules. ■ IDLE - Performing no work because there are no LCRs to send to the XStream client application. ■ BROWSING LCR - Browsing the outbound server's queue for the next LCR. ■ EVALUATING RULES - Evaluating an LCR against a rule set. ■ DEQUEUING LCR - Dequeuing an LCR from the outbound server's queue. ■ SENDING LCR - Sending an LCR to an XStream client application. ■ WAITING FOR CAPTURE TO TERMINATE - Waiting for the capture process to become disabled. ■ SUSPENDED DUE TO A DROPPED SUBSCRIBER - Suspended because a connected subscriber was dropped. For example, a subscriber can be dropped during a split or merge operation. ■ SUSPENDED FOR AUTO SPLIT/MERGE - Suspended because an automatic split or merge operation is being performed. ■ WAITING ON EMPTY QUEUE - Waiting for more LCRs from the capture process. ■ WAITING FOR CLIENT - Waiting for the XStream client application to request more LCRs. ■ WAITING FOR CAPTURE TO INITIALIZE - Waiting for the capture process to finish the data dictionary build. ■ WAITING TO ATTACH TO CAPTURE - Waiting for the outbound server to attach to the capture process. When a state refers to a capture process, it is the capture process that captures changes for the outbound server. When a state refers to a propagation, it is the outbound server that sends LCRs to the XStream client application.
XIDUSN	NUMBER	Transaction ID undo segment number of the transaction currently being processed This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
XIDSLT	NUMBER	Transaction ID slot number of the transaction currently being processed This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.

Column	Data Type	Description
XIDSQN	NUMBER	Transaction ID sequence number of the transaction currently being processed This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
COMMITSCN	NUMBER	Commit SCN of the transaction currently being processed This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
TOTAL_TRANSACTIONS_SENT	NUMBER	Total number of transactions sent by the outbound server to the XStream client application since the last time the client application attached to the outbound server This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
MESSAGE_SEQUENCE	NUMBER	Number of the current LCR being processed by the outbound server. This value is reset to 1 at the beginning of each transaction. This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
TOTAL_MESSAGES_SENT	NUMBER	Total number of LCRs sent by the outbound server to the XStream client application since the last time the client application attached to the outbound server
SEND_TIME	DATE	Time the last LCR was sent by the outbound server to the XStream client application
LAST_SENT_MESSAGE_NUMBER	NUMBER	Message number of the last LCR sent by the outbound server to the XStream client application
LAST_SENT_MESSAGE_CREATE_TIME	DATE	Creation time at the source database of the last LCR sent by the outbound server to the client application
ELAPSED_SEND_TIME	NUMBER	Time elapsed (in hundredths of a second) sending LCRs to the XStream client application since the last time the client application attached to the outbound server
COMMIT_POSITION	RAW(64)	Commit position of the transaction currently being processed This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
LAST_SENT_POSITION	RAW(64)	Position of the last LCR sent to the XStream client application This column is populated only if the COMMITTED_DATA_ONLY column shows YES. When the COMMITTED_DATA_ONLY column shows NO, this column is NULL.
BYTES_SENT	NUMBER	Total number of bytes sent by the outbound server to the XStream client application since the last time the client application attached to the outbound server
COMMITTED_DATA_ONLY	VARCHAR2(3)	YES if the outbound server can send only LCRs in committed transactions to the XStream client application. A committed transaction is an assembled, noninterleaving transaction with no rollbacks. NO if the outbound server can send LCRs in transactions that have not yet committed to the XStream client application. This mode is for internal Oracle use only.

V$XSTREAM_TRANSACTION

Note: This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

`V$XSTREAM_TRANSACTION` displays information about transactions that are being processed by capture processes, outbound servers, and inbound servers. This view can identify long running transactions and display how many LCRs are being processed in each transaction. This view only contains information about captured LCRs. It does not contain information about user-enqueued LCRs or user messages.

This view only shows information about LCRs that are being processed because they satisfied the rule sets for the component at the time of the query. For capture processes, this view only shows information about changes in transactions that the capture process has converted into LCRs. It does not show information about all the active transactions present in the redo log.

For outbound servers, this view only shows information about LCRs that the outbound server has dequeued. It does not show information about LCRs in the outbound server's queue. For outbound servers, information about a transaction remains in the view until the transaction is sent to the XStream client application.

For inbound servers, information about a transaction remains in the view until the transaction commits or until the entire transaction is rolled back.

Note: This view does not display information about Oracle Streams transactions. To view information about Oracle Streams transactions, query the `V$STREAMS_TRANSACTION` view.

Column	Data Type	Description
`COMPONENT_NAME`	`VARCHAR2(30)`	Name of the component
`COMPONENT_TYPE`	`VARCHAR2(10)`	Type of component: ■ `CAPTURE` for a capture process ■ `APPLY` for the apply reader subcomponent in an outbound server or inbound server ■ `PROPAGATION_SENDER` for the propagation sender that sends LCRs from a capture process to an outbound server
`XIDUSN`	`NUMBER`	Transaction ID undo segment number of the transaction
`XIDSLT`	`NUMBER`	Transaction ID slot number of the transaction
`XIDSQN`	`NUMBER`	Transaction ID sequence number of the transaction
`CUMULATIVE_MESSAGE_COUNT`	`NUMBER`	Number of LCRs processed in the transaction. If a component is restarted while the transaction is being processed, then this column shows the number of LCRs processed in the transaction since the component was started.
`TOTAL_MESSAGE_COUNT`	`NUMBER`	Total number of LCRs processed in the transaction by an outbound server or inbound server. This column does not pertain to capture processes.
`FIRST_MESSAGE_TIME`	`DATE`	Time stamp of the first LCR processed in the transaction. If a capture process is restarted while the transaction is being processed, then this column shows the time stamp of the first LCR processed after the capture process was started.
`FIRST_MESSAGE_NUMBER`	`NUMBER`	SCN of the first message in the transaction. If a capture process is restarted while the transaction is being processed, then this column shows the SCN of the first message processed after the capture process was started.
`LAST_MESSAGE_TIME`	`DATE`	Time stamp of the last LCR processed in the transaction
`LAST_MESSAGE_NUMBER`	`NUMBER`	SCN of the most recent message encountered in the transaction
`FIRST_MESSAGE_POSITION`	`RAW(64)`	Position of the first message seen by an XStream inbound server This column is populated only for an apply process that is functioning as an XStream inbound server.

Column	Data Type	Description
LAST_MESSAGE_POSITION	RAW(64)	Position of the last message seen by an XStream inbound server This column is populated only for an apply process that is functioning as an XStream inbound server.
TRANSACTION_ID	VARCHAR2(128)	Transaction ID for an XStream inbound server This column is populated only for an apply process that is functioning as an XStream inbound server.

Index