Oracle® Database

XStream Guide

23ai

F47495-04

March 2025

Oracle Database XStream Guide, 23ai

F47495-04

Primary Authors: Sunil Surabhi, Roopesh Ashok Kumar

Contributors: Prakash Jashnani, Alan Downing, Thuvan Hoang, Richard Huang, Joydip Kundu, Belinda Leung, Tianshu Li, Edwina Lu, Rui Mao, Pat McElroy, Valarie Moore, Srikanth Nalla, Partha Raghunathan, Ashish Ray, Jim Stamos, Byron Wang, Rod Ward, Lik Wong, Haobo Xu, Kevin Xu, Jun Yuan, Lei Zheng, Volker Kuhr, Jing Liu, Lewis Kaplan, Hung Tran, Mahesh Subramaniam, Vincent Gerard, Fernando Gutierrez Mendez, Qinqin Wang, Jorge Rivera, Susana Garduno, Roberto Morales

Contents

Part IV Appendixes

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, 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 that have purchased support 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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

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 General Concepts and Use Cases

Database administrators who configure and manage XStream environments must understand XStream concepts and use cases.

- Introduction to XStream
  XStream enables information sharing with outstanding performance and usability.
- General XStream Concepts
  General XStream concepts apply to both XStream Out and XStream In.

1 Introduction to XStream

XStream enables information sharing with outstanding performance and usability.

- 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.
- Purpose of XStream
  Some customers, especially Independent Software Vendors (ISVs) and partners, need access to the Oracle database as a platform for their own information sharing products, such as file-level replication, middle-tier caches or even to support replication between Oracle and non-Oracle data stores. XStream provides these customers with fast, real-time access to changes made in the Oracle database.
- XStream Use Cases
  There are several common XStream use cases.
- Prerequisites for XStream
  Meet prerequisites before using XStream.
- XStream Security Model
  To use the XStream APIs and manage XStream configurations, you must be granted the roles `XSTREAM_CAPTURE` or `XSTREAM_APPLY`.
- Tasks and Tools for XStream
  You perform common tasks with XStream and use a set of tools to complete these tasks.

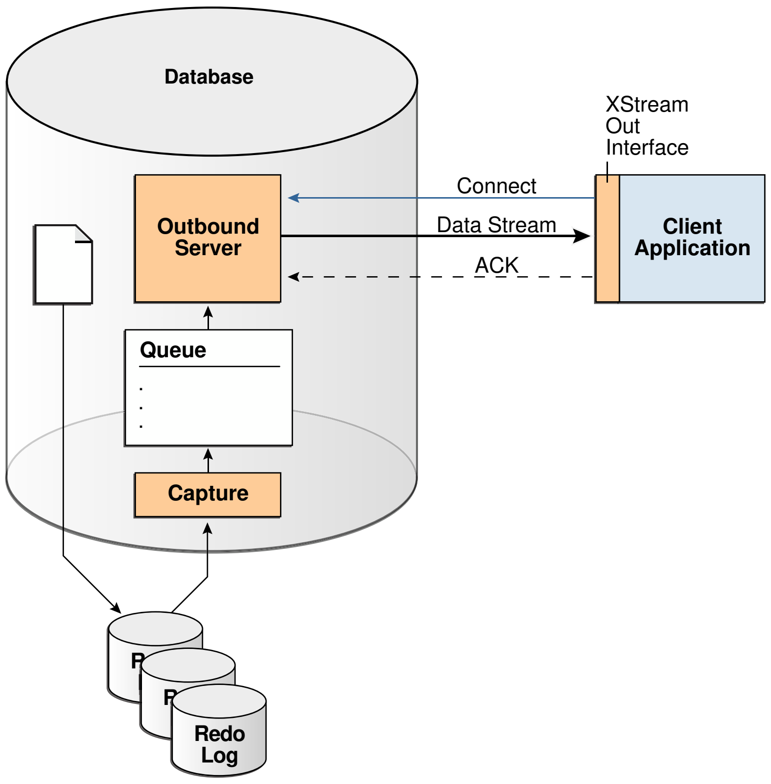
About XStream

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.

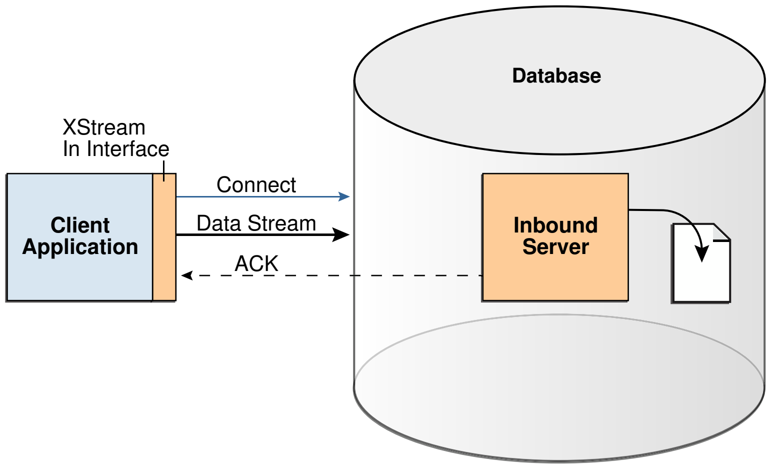
XStream consists of two major features: XStream Out and XStream In. **XStream Out** provides Oracle Database components and APIs that enable you to share data changes made to an Oracle database with other systems. XStream Out can retrieve both data manipulation language (DML) and data definition language (DDL) changes from the redo log and send these changes to a client application that uses the APIs, as shown in the following figure.

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. XStream In can apply these changes to database objects in the Oracle database, as shown in the following figure.

Figure 1-2 XStream In



XStream uses the capture and apply features of the Oracle database. These features enable the following functionality for XStream:

- 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.

- Rules and rule sets that control behavior, including inclusion and exclusion rules

  Rules enable the filtering of database changes at the database level, schema level, table level, and row/column level.
- Rule-based transformations that modify captured data changes
- Support for most data types in the database, including LOBs, `LONG`, `LONG RAW`, `JSON`, `BOOLEAN`, 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:

In both XStream Out and XStream In configurations, the client application must use a dedicated server connection.

Related Topics

- Configuring XStream In
  You can configure the Oracle Database components that are used by XStream.
- Logical Change Records (LCRs)
  An LCR is a message with a specific format that describes a database change.
- Rules and Rule Sets
  XStream uses rules and rule sets.
- *Oracle Database PL/SQL Packages and Types Reference*

Purpose of XStream

Some customers, especially Independent Software Vendors (ISVs) and partners, need access to the Oracle database as a platform for their own information sharing products, such as file-level replication, middle-tier caches or even to support replication between Oracle and non-Oracle data stores. XStream provides these customers with fast, real-time access to changes made in the Oracle database.

XStream is a programmatic interface that allows client applications to connect to the Oracle database and attach directly into the database capture or apply process. A client application can take advantage of a high performing capture mechanism by attaching to the XStream outbound server to directly access the stream of changes from an Oracle database. XStream Out streams logical change records (LCRs) to the client application in committed transaction order.

To apply changes to the Oracle database, a client application can hook directly into the XStream inbound server. The application provides the inbound server with LCRs in transactional order and can take advantage of the high performance of the database apply engine to apply the changes to the Oracle database.

XStream Use Cases

There are several common XStream use cases.

XStream provides a flexible infrastructure for sharing information between Oracle data sources and non-Oracle data sources. You can use XStream to meet the data and informational sharing needs of various organizations.

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

- A capture process captures data changes made to an Oracle database.
- XStream Out 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.

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.

- Replicating Data Changes with Non-Oracle Databases
  Replication is generally used to improve availability and to improve performance by spreading the network load over multiple regions and servers.
- Using Files to Store Data Changes
  Some environments use files to store data changes.
- Sharing Data Changes with a Client-Side Memory Cache
  Some environments cache data in memory to improve performance.

See Also:

- "Introduction to XStream Out"
- "Introduction to XStream In"

Replicating Data Changes with Non-Oracle Databases

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.

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.

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.

Note:

Oracle GoldenGate is a complete solution for replicating data between Oracle and non-Oracle databases. Oracle GoldenGate documentation for more information.

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"

- 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.

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.

Prerequisites for XStream

Meet prerequisites before using XStream.

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 Also:

  Oracle Database Concepts for information about this topic

- Knowledge of distributed databases

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

  See Also:

  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 Also:

  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 Also:

  – *Oracle Call Interface Developer's Guide* for information about the OCI API

  – *Oracle Database Get Started with Java Development* and *Oracle Database Java Developer's Guide* for information about the Java API

Note:

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

XStream Security Model

To use the XStream APIs and manage XStream configurations, you must be granted the roles `XSTREAM_CAPTURE` or `XSTREAM_APPLY`.

The role required by the user depends on the type of configuration they manage, XStream Out or XStream In. Each user can only manage the XStream components that they own.

See Also :

- "XStream Out and Security"
- "XStream In and Security"
- "Configure an XStream Administrator on All Databases"
- "Configure an XStream Administrator"

Tasks and Tools for XStream

You perform common tasks with XStream and use a set of tools to complete these tasks.

- XStream Tasks
  You complete common tasks with XStream.
- XStream Tools
  You use a set of tools to complete tasks with XStream.

XStream Tasks

You complete common tasks with XStream.

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 Also:

  Configuring XStream Out and Configuring XStream In 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 Also:**
>
> Managing XStream Out and Managing XStream In for information about this task

- Monitor XStream

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

  > **See Also:**
  >
  > The Oracle Enterprise Manager Cloud Control online help, Monitoring XStream Out and Monitoring XStream In for information about this task

XStream Tools

You use a set of tools to complete tasks with XStream.

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 Also:**
  >
  > *Oracle Database SQL Language Reference*, *Oracle Database Reference*, *Oracle Database PL/SQL Language Reference*, and *Oracle Database PL/SQL Packages and Types Reference* for information about this topic

- Oracle Enterprise Manager Cloud Control

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

  See the Oracle Enterprise Manager Cloud Control 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 Also:

- *Oracle Call Interface Developer's Guide* for information about the OCI API
- *Oracle Database XStream Java API Reference* for information about the XStream Java API
- *Oracle Database Get Started with Java Development* and *Oracle Database Java Developer’s Guide* for information about the Java API

2

General XStream Concepts

General XStream concepts apply to both XStream Out and XStream In.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- Logical Change Records (LCRs)
  An LCR is a message with a specific format that describes a database change.
- Rules and Rule Sets
  XStream uses rules and rule sets.
- Rule-Based Transformations
  In XStream, a rule-based transformation is any modification to a logical change record (LCR) when a rule in a positive rule set evaluates to `TRUE`.
- XStream and the Oracle Replication Performance Advisor
  The Oracle Replication Performance Advisor consists a collection of data dictionary views.
- Using Automatic Workload Repository (AWR) Reports for Oracle Database
- Automatic Workload Repository (AWR) Report for XStream
  Automatic Workload Repository (AWR) reports help leverage the existing statistics from AWR tables.
- XStream and SQL Generation
  SQL generation is the ability to generate the SQL statement required to perform the change encapsulated in a row LCR.

See Also:

- XStream Out Concepts
- XStream In Concepts

Logical Change Records (LCRs)

An LCR is a message with a specific format that describes a database change.

There are three types of LCRs: row LCRs, DDL LCRs, and sequence LCRs. In XStream, an LCR is the basic unit of information that describes a database change.

In an XStream Out configuration, a capture process can capture LCRs and send them to an outbound server. The outbound server can send the LCRs to the XStream client application.

In an XStream In configuration, an XStream client application can construct LCRs and send them to an inbound server. The inbound server can apply the database changes directly to the database object in the database, or the inbound server can process the LCRs in a customized way.

- Row LCRs
  A row LCR describes a change to the data in a single row or a change to a single LOB column, `LONG` column, `LONG RAW` column, or `XMLType` column in a row.
- DDL LCRs
  A DDL LCR describes a data definition language (DDL) change.
- Extra Information in Row LCRs and DDL LCRs
  In addition to the information discussed in the previous sections, row LCRs and DDL LCRs optionally can include extra information (or LCR attributes).
- Sequence LCRs
  A sequence LCR is a row LCR that includes information about sequence values. Sequence database objects generate sequence values.
- Position Order in an LCR Stream
  Each LCR has a position attribute. The position of an LCR identifies its placement in the stream of LCRs in a transaction.
- LCRIDs and the Position of LCRs
  An LCRID is the raw value that specifies the position of an LCR for XStream Out. It is strictly increasing, uniquely identifies an LCR, and is persistent across restart. XStream uses LCRID values for ordering logical change records (LCRs) and for determining which LCRs and transactions have been received and applied.

Row LCRs

A row LCR describes a change to the data in a single row or a change to a single LOB column, `LONG` column, `LONG RAW` column, or `XMLType` column in a row.

The change results from a data manipulation language (DML) statement or a piecewise operation. It may help to think of a row LCR as a DML LCR. For example, a single DML statement can insert or merge multiple rows into a table, can update multiple rows in a table, or can delete multiple rows from a table.

Since a single DML statement can affect more than one row, the capture process creates a row LCR for each row that is changed by the DML statement. Row LCRs represent the data changes made by a SQL or PL/SQL procedure invocation.

Each row LCR is encapsulated in an object of `LCR$_ROW_RECORD` type. The following table describes the attributes that are present in each row LCR.

Table 2-1 Attributes Present in All Row LCRs

Attribute	Description
`source_database_name`	The name of the source database where the row change occurred. If the LCRs originated in a multitenant container database (CDB), then this attribute specifies the global name container where the row change occurred.
`command_type`	The type of DML statement that produced the change, either `INSERT`, `UPDATE`, `DELETE`, `LOB ERASE`, `LOB WRITE`, or `LOB TRIM`.

Table 2-1 (Cont.) Attributes Present in All Row LCRs

Attribute	Description
`object_owner`	The schema name that contains the table with the changed row.
`object_name`	The name of the table that contains the changed row.
`tag`	A raw tag that you can use to track the LCR.
`transaction_id`	The identifier of the transaction in which the DML statement was run.
`scn`	The system change number (SCN) at the time when the change was made.
`old_values`	The old column values related to the change. These are the column values for the row before the DML change. If the type of the DML statement is `UPDATE` or `DELETE`, then these old values include some or all of the columns in the changed row before the DML statement. If the type of the DML statement is `INSERT`, then there are no old values. For `UPDATE` and `DELETE` statements, row LCRs created by a capture process can include some or all of the old column values in the row.
`new_values`	The new column values related to the change. These are the column values for the row after the DML change. If the type of the DML statement is `UPDATE` or `INSERT`, then these new values include some or all of the columns in the changed row after the DML statement. If the type of the DML statement is `DELETE`, then there are no new values. For `UPDATE` and `INSERT` statements, row LCRs created by a capture process can include some or all of the new column values in the row.
`position`	A unique identifier of `RAW` data type for each LCR. The position is strictly increasing within a transaction and across transactions. LCR position is commonly used in XStream configurations. See "Position Order in an LCR Stream".
`root_name`	If the LCR originated in a CDB, then this attribute specifies the global name of the root in the CDB. If the LCR originated in a non-CDB, then this attribute is the same as the `source_database_name` attribute.

Row LCRs that were captured by a capture process in an XStream Out configuration contain additional attributes. The following table describes these additional attributes. These attributes are not present in row LCRs constructed by an XStream client application in an XStream In configuration.

Table 2-2 Additional Attributes in LCRs Captured by a Capture Process

Attribute	Description
`commit_scn`	The commit system change number (SCN) of the transaction to which the LCR belongs.
`commit_scn_from_position`	The commit system change number (SCN) of a transaction determined by the input position, which is generated by an XStream outbound server.
`commit_time`	The commit time of the transaction to which the LCR belongs.
`compatible`	The minimal database compatibility required to support the LCR.
`instance_number`	The instance number of the database instance that made the change that is encapsulated in the LCR. Typically, the instance number is relevant in an Oracle Real Application Clusters (Oracle RAC) configuration.
`lob_information`	The LOB information for the column, such as `NOT_A_LOB` or `LOB_CHUNK`.
`lob_offset`	The LOB offset for the specified column in the number of characters for `CLOB` columns and the number of bytes for `BLOB` columns.

Table 2-2 (Cont.) Additional Attributes in LCRs Captured by a Capture Process

Attribute	Description
`lob_operation_size`	The operation size for the LOB column in the number of characters for `CLOB` columns and the number of bytes for `BLOB` columns.
`long_information`	The `LONG` information for the column, such as `NOT_A_LONG` or `LONG_CHUNK`.
`row_text`	The SQL statement for the change that is encapsulated in the row LCR.
`scn_from_position`	The SCN of the LCR.
`source_time`	The time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.
`xml_information`	The XML information for the column, such as `NOT_XML`, `XML_DOC`, or `XML_DIFF`.

- Row LCR Subtypes
  A row LCR can also contain transaction control statements. These row LCRs contain transaction control directives such as `COMMIT` and `ROLLBACK`.

Row LCR Subtypes

A row LCR can also contain transaction control statements. These row LCRs contain transaction control directives such as `COMMIT` and `ROLLBACK`.

Such row LCRs are internal and can be used by outbound servers, inbound servers, and XStream client applications to maintain transaction consistency.

See Also:

Oracle Database PL/SQL Packages and Types Reference

DDL LCRs

A DDL LCR describes a data definition language (DDL) change.

A DDL statement changes the structure of the database. For example, a DDL statement can create, alter, or drop a database object.

Each DDL LCR is encapsulated in an object of `LCR$_DDL_RECORD` type. The following table describes the attributes that are present in each DDL LCR.

Table 2-3 Attributes Present in All DDL LCRs

Attribute	Description
`source_database_name`	The name of the source database where the DDL change occurred. If the LCRs originated in a CDB, then this attribute specifies the global name of the container where the DDL change occurred.
`command_type`	The type of DDL statement that produced the change, for example `ALTER TABLE` or `CREATE INDEX`.

Table 2-3 (Cont.) Attributes Present in All DDL LCRs

Attribute	Description
`object_owner`	The schema name of the user who owns the database object on which the DDL statement was run.
`object_name`	The name of the database object on which the DDL statement was run.
`object_type`	The type of database object on which the DDL statement was run, for example `TABLE` or `PACKAGE`.
`ddl_text`	The text of the DDL statement.
`logon_user`	The logon user, which is the user whose session executed the DDL statement.
`current_schema`	The schema that is used if no schema is specified for an object in the DDL text.
`base_table_owner`	The base table owner. If the DDL statement is dependent on a table, then the base table owner is the owner of the table on which it is dependent.
`base_table_name`	The base table name. If the DDL statement is dependent on a table, then the base table name is the name of the table on which it is dependent.
`tag`	A raw tag that you can use to track the LCR.
`transaction_id`	The identifier of the transaction in which the DDL statement was run.
`scn`	The system change number (SCN) at the time when the change was made.
`position`	A unique identifier of `RAW` data type for each LCR. The position is strictly increasing within a transaction and across transactions. LCR position is commonly used in XStream configurations. See "Position Order in an LCR Stream".
`edition_name`	The name of the edition in which the DDL statement was executed.
`root_name`	If the LCR originated in a CDB, then this attribute specifies the global name of the root in the CDB. If the LCR originated in a non-CDB, then this attribute is the same as the `source_database_name` attribute.

DDL LCRs that were captured by a capture process contain additional attributes. The following table describes these additional attributes. These attributes are not present in DDL LCRs constructed by an XStream client application in an XStream In configuration.

Table 2-4 Additional Attributes in DDL LCRs Captured by a Capture Process

Attribute	Description
`commit_scn`	The commit system change number (SCN) of the transaction to which the LCR belongs.
`commit_scn_from_position`	The commit SCN of a transaction determined by the input position, which is generated by an XStream outbound server.
`commit_time`	The commit time of the transaction to which the LCR belongs.
`compatible`	The minimal database compatibility required to support the LCR.
`instance_number`	The instance number of the database instance that made the change that is encapsulated in the LCR. Typically, the instance number is relevant in an Oracle Real Application Clusters (Oracle RAC) configuration.
`scn_from_position`	The SCN of the LCR.
`source_time`	The time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.

Note:

Both row LCRs and DDL LCRs contain the source database name of the database where a change originated. To avoid problems, Oracle recommends that you do not change the global name of the source database after a capture process has started capturing changes.

See Also:

- *Oracle Call Interface Developer's Guide* for a complete list of the types of DDL statements in the "SQL Command Codes" table
- *Oracle Database PL/SQL Packages and Types Reference*

Extra Information in Row LCRs and DDL LCRs

In addition to the information discussed in the previous sections, row LCRs and DDL LCRs optionally can include extra information (or LCR attributes).

The extra attributes in LCRs are described in the following table.

Table 2-5 Extra Attributes in LCRs

Attribute	Description
`row_id`	The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs or row LCRs for index-organized tables.
`serial#`	The serial number of the session that performed the change captured in the LCR.
`session#`	The identifier of the session that performed the change captured in the LCR.
`thread#`	The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment.
`tx_name`	The name of the transaction that includes the LCR.
`username`	The name of the current user who performed the change captured in the LCR.

You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a capture process to capture one or more extra attributes.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `INCLUDE_EXTRA_ATTRIBUTE` procedure
- *Oracle Database PL/SQL Language Reference* for more information about the current user

Sequence LCRs

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 XStream inbound server or an Oracle Apply process 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 inbound server or apply process to use sequence LCRs, set the `apply_sequence_nextval` apply parameter to `Y`.

Note:

Sequence LCRs are intended for one-way replication configurations. Sequence LCRs cannot be used in bidirectional replication configurations.

See Also:

- "Setting a Capture Process Parameter"
- *Oracle Call Interface Developer's Guide* 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

Position Order in an LCR Stream

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

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 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.

See Also:

- "Position of LCRs and XStream Out"
- "Position of LCRs and XStream In"

LCRIDs and the Position of LCRs

An LCRID is the raw value that specifies the position of an LCR for XStream Out. It is strictly increasing, uniquely identifies an LCR, and is persistent across restart. XStream uses LCRID values for ordering logical change records (LCRs) and for determining which LCRs and transactions have been received and applied.

Starting with Oracle Database 12*c* Release 2 (12.2.0.1), the LCRID is versioned. When you create or add an outbound server, you can choose the LCRID version it uses. To specify version 2, the database compatibility level must be at 12.2.0 or higher. By default, an outbound server created or added when database compatibility is lower than 12.2.0 uses LCRID version 1, and an outbound server created or added when database compatibility is at 12.2.0 or higher uses LCRID version 2. You might choose to use LCRID version 1 for an outbound server if, for example, the outbound server captures LCRs that will be applied at a database that is at a lower compatibility level.

After an outbound server is created or added, its LCRID version cannot be changed. To change the LCRID version, you must drop and re-create the outbound server. If the outbound server was sending LCRs to an inbound server, then you must drop and re-create the inbound server.

The same database change has different LCRID values for version 1 and version 2. New functions in the `DBMS_XSTREAM_ADM` package enable you to compare any stored LCRID values in different versions and convert LCRID values from one version to another. Specifically, the `COMPARE_POSITION` function compares two LCRID values, and the `CONVERT_POSITION` function converts LCRID values from one version to another.

Related Topics

- Configuring XStream Out
  An outbound server in an XStream Out configuration streams Oracle database changes to a client application.

Rules and Rule Sets

XStream uses rules and rule sets.

- Rules and Rule Sets Defined
  A rule is a database object that enables a client to perform an action when an event occurs and a condition is satisfied. In an XStream configuration, rules identify which LCRs to stream from one component to another.
- Rule Sets and XStream Components
  An XStream component performs its task if a database change satisfies its rule sets.
- System-Created Rules and XStream
  An XStream component performs its task for an LCR if the LCR satisfies its rule sets. A system-created rule is created by the `DBMS_XSTREAM_ADM` package.

See Also:

- "Managing Rules for an XStream Out Configuration"
- "Monitoring XStream Rules"

Rules and Rule Sets Defined

A rule is a database object that enables a client to perform an action when an event occurs and a condition is satisfied. In an XStream configuration, rules identify which LCRs to stream from one component to another.

Capture processes, propagations, outbound servers and inbound servers can use rules. You can configure rules for each XStream component independently, and the rules for different XStream components do not need to match.

A rule set is a collection of rules. The behavior of each XStream component is determined by the rules in the rule sets that are associated with it. You can associate a positive rule set and a negative rule set with each XStream component.

In addition, a single rule pertains to either the results of data manipulation language (DML) changes or data definition language (DDL) changes. So, for example, you must use at least two rules to include all of the changes to a particular table: one rule for the results of DML changes and another rule for DDL changes.

The results of a DML change are row changes, and an LCR that encapsulates a row change is called a row LCR. A single DML change can result in multiple row changes. Therefore, a single DML change can result in multiple row LCRs. An LCR that encapsulates a DDL change is called a DDL LCR.

Rule Sets and XStream Components

An XStream component performs its task if a database change satisfies its rule sets.

In general, a change satisfies the rule sets when *no rules* in the negative rule set evaluate to `TRUE` for the change and *at least one rule* in the positive rule set evaluates to `TRUE` for the change. The negative rule set is always evaluated first.

You use rule sets in an XStream configuration to specify the following:

- Changes that a capture process captures from the redo log or discards. If a change found in the redo log satisfies the rule sets for a capture process, then the capture process captures the change. If a change found in the redo log does not satisfy the rule sets for a capture process, then the capture process discards the change.

  In XStream Out configurations that share one capture process among several outbound servers, the rules for the capture process must pass the LCRs that are needed by any of the outbound servers that share the capture process.
- The LCRs that a propagation sends from one queue to another or discards. If an LCR in a queue satisfies the rule sets for a propagation, then the propagation sends the LCR. If an LCR in a queue does not satisfy the rule sets for a propagation, then the propagation discards the LCR.
- The LCRs that an outbound server sends to an XStream client application or discards. If an LCR satisfies the rule sets for an outbound server, then the outbound server sends the LCR to the XStream client application. If an LCR does not satisfy the rule sets for an outbound server, then the outbound server discards the LCR.
- The LCRs that an inbound server applies or discards. If an LCR satisfies the rule sets for an inbound server, then the inbound server applies the LCR. If an LCR in not satisfy the rule sets for an inbound server, then the inbound server discards the LCR.

When an XStream component has no rule sets, the component performs its task for all database changes. For example, if an inbound server has no rule sets, then it applies all of the LCRs sent to it by an XStream client application.

System-Created Rules and XStream

An XStream component performs its task for an LCR if the LCR satisfies its rule sets. A system-created rule is created by the `DBMS_XSTREAM_ADM` package.

A system-created rule can specify one of the following levels of granularity: table, schema, or global.

- XStream System-Created Rule Procedures
  Several PL/SQL procedures in the `DBMS_XSTREAM_ADM` package can create system-generated rules.

- Global Rules
  When you use a rule to specify a task that is relevant to an entire database, you are specifying a global rule.
- Schema Rules
  When you use a rule to specify a task that is relevant to a schema, you are specifying a schema rule.
- Table Rules
  When you use a rule to specify a task that is relevant to a table, you are specifying a table rule.
- Subset Rules
  A subset rule is a special type of table rule for DML changes that is relevant only to a subset of the rows in a table.
- System-Created Rules and a Multitenant Environment
  A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.

See Also:

- "Managing Rules for an XStream Out Configuration"
- "Monitoring XStream Rules"
- *Oracle Database PL/SQL Packages and Types Reference*

XStream System-Created Rule Procedures

Several PL/SQL procedures in the `DBMS_XSTREAM_ADM` package can create system-generated rules.

There are three types of procedures that create system-created rules:

- Procedures that create or alter an outbound server and the rules for the outbound server

  These procedures include `CREATE_OUTBOUND`, `ADD_OUTBOUND`, and `ALTER_OUTBOUND`. These procedures make it easy to configure XStream Out quickly. If they meet your needs, then you should use these procedures to simplify XStream Out configuration. The `CREATE_OUTBOUND` procedure creates the queue and capture process used by the outbound server in addition to the outbound server.
- Procedures that create a propagation or add rules to an existing propagation

  These procedures include the `ADD_*_PROPAGATION_RULES` procedures. If the specified propagation does not exist, then these procedures create the propagation and add rules to the propagation's rule sets. If the specified propagation exists, then these procedures add rules to the existing propagation's rule sets.
- Procedures that add rules to an existing XStream component, such as a capture process, outbound server, or inbound server

  These procedures include the `ADD_*_RULES` procedures. These procedure provide more flexibility and fine-grained control over the system-created rules. You should use these procedures when necessary to add rules to your XStream configuration.

The following table describes which procedures can create rules for which XStream components.

Table 2-6 XStream System-Created Rule Procedures

Procedure	Capture Process	Propagation	Outbound Server	Inbound Server
`CREATE_OUTBOUND`	Yes	No	Yes	No
`ADD_OUTBOUND`	No	No	Yes	No
`ALTER_OUTBOUND`	Yes	No	Yes	No
`ADD_GLOBAL_RULES`	Yes	No	Yes	Yes
`ADD_GLOBAL_PROPAGATION_RULES`	No	Yes	No	No
`ADD_SCHEMA_RULES`	Yes	No	Yes	Yes
`ADD_SCHEMA_PROPAGATION_RULES`	No	Yes	No	No
`ADD_GLOBAL_RULES`	Yes	No	Yes	Yes
`ADD_SUBSET_OUTBOUND_RULES`	No	No	Yes	No
`ADD_SUBSET_RULES`	Yes	No	Yes	Yes
`ADD_SUBSET_PROPAGATION_RULES`	No	Yes	No	No
`ADD_TABLE_RULES`	Yes	No	Yes	Yes
`ADD_TABLE_PROPAGATION_RULES`	No	Yes	No	No

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about these procedures

Global Rules

When you use a rule to specify a task that is relevant to an entire database, you are specifying a global rule.

You can specify a global rule for DML changes, a global rule for DDL changes, or a global rule for each type of change (two rules total).

A single global rule in the positive rule set for a capture process means that the capture process captures the results of either all DML changes or all DDL changes to the source database. A single global rule in the negative rule set for a capture process means that the capture process discards the results of either all DML changes or all DDL changes to the source database.

A single global rule in the positive rule set for a propagation means that the propagation rule controls the set of LCRs that are applicable to a specific outbound server. If a single capture services multiple outbound servers, the set of changes distributed to each outbound server is controlled by the propagation rules (the capture rules are the superset of all changes). A single global rule in the negative rule set for a propagation means that the propagation discards either all row LCRs or all DDL LCRs from the capture process.

A single global rule in the positive rule set for an outbound server means that the outbound server sends either all row LCRs or all DDL LCRs that it receives to an XStream client application. A single global rule in the negative rule set for an outbound server means that the outbound server discards either all row LCRs or all DDL LCRs that it receives.

A single global rule in the positive rule set for an inbound server means that the inbound server applies either all row LCRs or all DDL LCRs sent to it by the XStream client application. A single global rule in the negative rule set for an inbound server means that the inbound server discards either all row LCRs or all DDL LCRs sent to it by the XStream client application.

When an inbound server should apply all of the LCRs it receives from its client application, you can configure the inbound server with no rule sets instead of using global rules. Also, for an inbound server to perform best, it should not receive LCRs that it should not apply.

To specify global rules for an outbound server, use the `ALTER_OUTBOUND` procedure or, for specifying a greater level of detail, the `ADD_GLOBAL_RULES` procedure in the `DBMS_XSTREAM_ADM` package.

To specify global rules for an inbound server, use the `ALTER_INBOUND` procedure or, for specifying a greater level of detail, the `ADD_GLOBAL_RULES` procedure in the `DBMS_XSTREAM_ADM` package.

See Also:

- "Managing Rules for an XStream Out Configuration"
- "Monitoring XStream Rules"
- *Oracle Database PL/SQL Packages and Types Reference*

Schema Rules

When you use a rule to specify a task that is relevant to a schema, you are specifying a schema rule.

You can specify a schema rule for DML changes, a schema rule for DDL changes, or a schema rule for each type of change to the schema (two rules total).

A single schema rule in the positive rule set for a capture process means that the capture process captures either the DML changes or the DDL changes to the schema. A single schema rule in the negative rule set for a capture process means that the capture process discards either the DML changes or the DDL changes to the schema.

A single schema rule in the positive rule set for a propagation means that the propagation propagates either the row LCRs or the DDL LCRs in the source queue that contain changes to the schema. A single schema rule in the negative rule set for a propagation means that the propagation discards either the row LCRs or the DDL LCRs in the source queue that contain changes to the schema.

A single schema rule in the positive rule set for an outbound server means that the outbound server sends either the row LCRs or the DDL LCRs that it receives that contain changes to the schema to an XStream client application. A single schema rule in the negative rule set for an outbound server means that the outbound server discards either the row LCRs or the DDL LCRs that it receives that contain changes to the schema.

A single schema rule in the positive rule set for an inbound server means that the inbound server applies either the row LCRs or the DDL LCRs that it receives from an XStream client application that contain changes to the schema. A single schema rule in the negative rule set for an inbound server means that the inbound server discards either the row LCRs or the DDL LCRs that it receives from an XStream client application that contain changes to the schema.

To specify schema rules for either an outbound server or an inbound server, use the `ALTER_OUTBOUND` procedure or the `ADD_SCHEMA_RULES` procedure in the `DBMS_XSTREAM_ADM` package.

See Also:

- "Managing Rules for an XStream Out Configuration"
- "Monitoring XStream Rules"
- *Oracle Database PL/SQL Packages and Types Reference*

Table Rules

When you use a rule to specify a task that is relevant to a table, you are specifying a table rule.

You can specify a table rule for DML changes, a table rule for DDL changes, or a table rule for each type of change to the table (two rules total).

A single table rule in the positive rule set for a capture process means that the capture process captures either the DML changes or the DDL changes to the table. A single table rule in the negative rule set for a capture process means that the capture process discards either the DML changes or the DDL changes to the table.

A single table rule in the positive rule set for a propagation means that the propagation propagates either the row LCRs or the DDL LCRs in the source queue that contain changes to the table. A single table rule in the negative rule set for a propagation means that the propagation discards either the row LCRs or the DDL LCRs in the source queue that contain changes to the table.

A single table rule in the positive rule set for an outbound server means that the outbound server sends either the row LCRs or the DDL LCRs that it receives that contain changes to the table to an XStream client application. A single table rule in the negative rule set for an outbound server means that the outbound server discards either the row LCRs or the DDL LCRs that it receives that contain changes to the table.

A single table rule in the positive rule set for an inbound server means that the inbound server applies either the row LCRs or the DDL LCRs that it receives from an XStream client application that contain changes to the table. A single table rule in the negative rule set for an inbound server means that the inbound server discards either the row LCRs or the DDL LCRs that it receives from an XStream client application that contain changes to the table.

To specify table rules for an outbound server or inbound server, use either the `ALTER_OUTBOUND` procedure or `ADD_TABLE_RULES` in the `DBMS_XSTREAM_ADM` package.

Subset Rules

A subset rule is a special type of table rule for DML changes that is relevant only to a subset of the rows in a table.

When you create a subset rule, you use a condition similar to a `WHERE` clause in a `SELECT` statement to specify the following:

- That a capture process only captures a subset of the row changes resulting from DML changes to a particular table
- That a propagation only propagates a subset of the row LCRs relating to a particular table
- That an outbound server only sends a subset of the row LCRs relating to a particular table to an XStream client application
- That an inbound server only applies a subset of the row LCRs relating to a particular table

Supplemental logging is required when you specify the following types of subset rules:

- Subset rules for a capture process
- Subset rules for a propagation that will propagate LCRs captured by a capture process
- Subset rules for an outbound server that will send LCRs captured by a capture process to an XStream client application

In any of these cases, an unconditional supplemental log group must be specified at the source database for all the columns in the subset condition. In some cases, when a subset rule is specified, an update can be converted to an insert, and, in these cases, supplemental information might be needed for some or all of the columns.

To specify subset rules for an outbound server, use the `ADD_SUBSET_OUTBOUND_RULES`, `ADD_SUBSET_RULES`, or the `REMOVE_SUBSET_OUTBOUND_RULES` procedures in the `DBMS_XSTREAM_ADM` package.

See Also:

- "If Required, Configure Supplemental Logging"
- "Adding Subset Rules to an Outbound Server's Positive Rule Set"
- "Removing Subset Rules from an Outbound Server's Positive Rule Set"
- "Monitoring XStream Rules"

System-Created Rules and a Multitenant Environment

A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.

It can also contain application containers. An application container is an optional component of a CDB that consists of an application root and the application PDBs associated with it. An application container stores data for one or more applications. An application container shares application metadata and common data. In a CDB, each of the following is a container: the CDB root, each PDB, each application root, and each application PDB.

In a CDB, LCRs can contain the global name of the container where the change originated in the `source_database_name` attribute and the global name of the CDB root in the `root_name` attribute. The rules for XStream components can consider these attributes.

- System-Created Rules in a CDB and XStream Out
  In a CDB, XStream Out must be configured in the CDB root. Therefore, the PL/SQL procedures in the `DBMS_XSTREAM_ADM` package that create system-created rules must be run in the CDB root while connected as a common user.
- System-Created Rules in a CDB and XStream In
  You can configure XStream In in the root or in any container in a CDB.

Related Topics

- *Oracle Multitenant Administrator's Guide*

System-Created Rules in a CDB and XStream Out

In a CDB, XStream Out must be configured in the CDB root. Therefore, the PL/SQL procedures in the `DBMS_XSTREAM_ADM` package that create system-created rules must be run in the CDB root while connected as a common user.

Excluding the procedures that create rules for propagations, the procedures that create system-created rules for XStream Out, such as the `ADD_GLOBAL_RULES` procedure, include the key parameters in the following table:

Table 2-7 Key Procedure Parameters for System-Created Rules in a CDB

Parameter	Description
`source_database`	The global name of the source database. In a CDB, specify the global name of the container to which the rules pertain. The container can be the CDB root, a PDB, an application root, or an application PDB. The following are examples: `mycdb.example.com` or `hrpdb.example.com`.
`source_root_name`	The global name of the CDB root in the source CDB. The following are examples: `mycdb.example.com`.
`source_container_name`	The short name of the source container. The container can be the CDB root, a PDB, an application root, or an application PDB. The following are examples: `CDB$ROOT` or `hrpdb`.

If you do not include the domain name when you specify `source_database` or `source_root_name`, then the procedure appends it to the name automatically. For example, if you specify `DBS1` and the domain is `.EXAMPLE.COM`, then the procedure specifies `DBS1.EXAMPLE.COM` automatically.

The combination of these key parameters determines which containers' changes XStream Out captures and streams to the client application, based on the rules generated by the procedures. Regardless of the settings for these parameters, system-generated rules can still limit the changes captured and streamed to specific schemas and tables.

Local capture means that a capture process runs on the source CDB. In a local capture configuration, the `source_root_name` parameter specifies the global name of the CDB root in the local CDB. If this parameter is `NULL`, then the global name of the CDB root in the local CDB is specified automatically. The resulting rules include a condition for the global name of the CDB root in the current CDB.

Downstream capture means that a capture process runs on a CDB other than the source CDB. In a downstream capture configuration, the `source_root_name` parameter must be non-`NULL`, and it must specify the global name of the CDB root in the remote source CDB. The resulting rules include a condition for the global name of the CDB root in the remote CDB. If this parameter is `NULL`, then local capture is assumed.

The following table describes the rule conditions for various `source_database` and `source_container_name` parameter settings in a local capture configuration.

Table 2-8 Local Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
`NULL`	`NULL`	XStream Out captures and streams changes made in any container in the local CDB, including the CDB root, all PDBs, all application roots, and all application PDBs.
non-`NULL`	`NULL`	XStream Out captures and streams changes made in the specified source container of the local CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. The `DBMS_XSTREAM_ADM` procedure queries the `CDB_PDBS` view and `CDB_PROPERTIES` view to determine the `source_container_name` value.
`NULL`	non-`NULL`	XStream Out captures and streams changes made in the specified source container of the local CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. The `DBMS_XSTREAM_ADM` procedure queries the `CDB_PDBS` view and `CDB_PROPERTIES` view to determine the `source_database` value.
non-`NULL`	non-`NULL`	XStream Out captures and streams changes made in the specified source container of the local CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. If the prefix of the `source_database` value is different from the `source_container_name` value, then the resulting rules include a condition for the `source_database` value, and an internal table maps the `source_database` value to the `source_container_name` value.

The following table describes the rule conditions for various `source_database` and `source_container_name` parameter settings in a downstream capture configuration.

Table 2-9 Downstream Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
`NULL`	`NULL`	XStream Out captures and streams changes made in any container in the remote source CDB, including the CDB root, all PDBs, all application roots, and all application PDBs.

Table 2-9 (Cont.) Downstream Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
non-`NULL`	`NULL`	XStream Out captures and streams changes made in the specified source container of the remote source CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. The `DBMS_XSTREAM_ADM` procedure derives the `source_container_name` value from the prefix of `source_database` value.
`NULL`	non-`NULL`	The `DBMS_XSTREAM_ADM` procedure raises an error.
non-`NULL`	non-`NULL`	XStream Out captures and streams changes made in the specified source container of the remote source CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. If the prefix of the `source_database` value is different from the `source_container_name` value, then the resulting rules include a condition for the `source_database` value, and an internal table maps the `source_database` value to the `source_container_name` value.

Related Topics

- Local Capture and Downstream Capture
  You can configure a capture process to run locally on a source database or remotely on a downstream database.
- *Oracle Database PL/SQL Packages and Types Reference*

System-Created Rules in a CDB and XStream In

You can configure XStream In in the root or in any container in a CDB.

Typically, an inbound server does not use rule sets or rules. Instead, it usually processes all LCRs that it receives from its client application. An inbound server can apply changes to the current container only. Therefore, if an inbound server is configured in the CDB root, then it can apply changes only to the CDB root. If an inbound server is configured in a PDB, then it can apply changes only to that PDB. If an inbound server is configured in an application root, then it can apply changes only to that application root, and if an inbound server is configured in an application PDB, then it can apply changes only to that application PDB.

Related Topics

- *Oracle Multitenant Administrator's Guide*

Rule-Based Transformations

In XStream, a rule-based transformation is any modification to a logical change record (LCR) when a rule in a positive rule set evaluates to `TRUE`.

In general, it is best for the client application to perform transformations of the data. If this is not possible, then the database can perform some simple transformations on DML LCRs.

- Declarative Rule-Based Transformations
  Declarative rule-based transformations cover a set of common transformation scenarios for row LCRs.
- Declarative Rule-Based Transformation Ordering
  The order in which different types of rule-based transformations is evaluated is important as results will vary.
- Evaluating Transformation Ordering
  You can evaluate transformation ordering.

Declarative Rule-Based Transformations

Declarative rule-based transformations cover a set of common transformation scenarios for row LCRs.

You specify (or declare) such a transformation using one of the following procedures in the `DBMS_XSTREAM_ADM` package:

- `ADD_COLUMN` either adds or removes a declarative transformation that adds a column to a row LCR.
- `DELETE_COLUMN` either adds or removes a declarative transformation that deletes a column from a row LCR.
- `KEEP_COLUMNS` either adds or removes a declarative transformation that keeps a list of columns in a row LCR. The transformation removes columns that are not in the list from the row LCR.
- `RENAME_COLUMN` either adds or removes a declarative transformation that renames a column in a row LCR.
- `RENAME_SCHEMA` either adds or removes a declarative transformation that renames the schema in a row LCR.
- `RENAME_TABLE` either adds or removes a declarative transformation that renames the table in a row LCR.

When you specify a declarative rule-based transformation, you specify the rule that is associated with it. When the specified rule evaluates to `TRUE` for a row LCR, XStream performs the declarative transformation internally on the row LCR, without invoking PL/SQL.

Declarative rule-based transformations provide the following advantages:

- Performance is improved because the transformations are run internally without using PL/SQL.
- Complexity is reduced because custom PL/SQL functions are not required.

Declarative rule-based transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run one of the procedures to add a declarative transformation. If a DDL rule is specified, then an error is raised.

Declarative Rule-Based Transformation Ordering

The order in which different types of rule-based transformations is evaluated is important as results will vary.

By default, Oracle Database performs declarative transformations in the following order when the rule evaluates to `TRUE`:

1. Keep columns

2. Delete column
3. Rename column
4. Add column
5. Rename table
6. Rename schema

The results of a declarative transformation are used in each subsequent declarative transformation. For example, suppose the following declarative transformations are specified for a single rule:

- Delete column address
- Add column address

Assuming column address exists in a row LCR, both declarative transformations should be performed in this case because the column address is deleted from the row LCR before column address is added back to the row LCR. The following table shows the transformation ordering for this example.

Step Number	Transformation Type	Transformation Details	Transformation Performed?
1	Keep columns	-	-
2	Delete column	Delete column `address` from row LCR	Yes
3	Rename column	-	-
4	Add column	Add column `address` to row LCR	Yes
5	Rename table	-	-
6	Rename schema	-	-

Another scenario might rename a table and then rename a schema. For example, suppose the following declarative transformations are specified for a single rule:

- Rename table `john.customers` to `sue.clients`
- Rename schema `sue` to `mary`

Notice that the rename table transformation also renames the schema for the table. In this case, both transformations should be performed and, after both transformations, the table name becomes `mary.clients`. The following table shows the transformation ordering for this example.

Step Number	Transformation Type	Transformation Details	Transformation Performed?
1	Keep columns	-	-
2	Delete column	-	-
3	Rename column	-	-
4	Add column	-	-
5	Rename table	Rename table `john.customers` to `sue.clients`	Yes
6	Rename schema	Rename schema `sue` to `mary`	Yes

Consider a similar scenario in which the following declarative transformations are specified for a single rule:

- Rename table `john.customers` to `sue.clients`
- Rename schema `john` to `mary`

In this case, the first transformation is performed, but the second one is not. After the first transformation, the table name is `sue.clients`. The second transformation is not performed because the schema of the table is now `sue`, not `john`. The following table shows the transformation ordering for this example.

Step Number	Transformation Type	Transformation Details	Transformation Performed?
1	Keep columns	-	-
2	Delete column	-	-
3	Rename column	-	-
4	Add column	-	-
5	Rename table	Rename table `john.customers` to `sue.clients`	Yes
6	Rename schema	Rename schema `john` to `mary`	No

The rename schema transformation is not performed, but it does not result in an error. In this case, the row LCR is transformed by the rename table transformation, and a row LCR with the table name `sue.clients` is returned.

Evaluating Transformation Ordering

You can evaluate transformation ordering.

- Row Migration Transformation Ordering
  In addition to declarative rule-based transformations, a row migration is an internal transformation that takes place when a subset rule evaluates to `TRUE`.
- User-Specified Declarative Transformation Ordering
  If you do not want to use the default declarative rule-based transformation ordering for a particular rule, then you can specify step numbers for each declarative transformation specified for the rule.
- Considerations for Rule-Based Transformations
  Several considerations apply to declarative rule-based transformations.

Row Migration Transformation Ordering

In addition to declarative rule-based transformations, a row migration is an internal transformation that takes place when a subset rule evaluates to `TRUE`.

You can use the `DBMS_XSTREAM_ADM.ADD_SUBSET_RULES` procedure to add subset rules. If both types of transformations are specified for a single rule, then Oracle Database performs the transformations in the following order when the rule evaluates to `TRUE`:

1. Row migration
2. Declarative rule-based transformation

User-Specified Declarative Transformation Ordering

If you do not want to use the default declarative rule-based transformation ordering for a particular rule, then you can specify step numbers for each declarative transformation specified for the rule.

If you specify a step number for one or more declarative transformations for a particular rule, then the declarative transformations for the rule behave in the following way:

- Declarative transformations are performed in order of increasing step number.
- The default step number for a declarative transformation is 0 (zero). A declarative transformation uses this default if no step number is explicitly specified for it.
- If two or more declarative transformations have the same step number, then these declarative transformations follow the default ordering described in "Declarative Rule-Based Transformation Ordering".

For example, you can reverse the default ordering for declarative transformations by specifying the following step numbers for transformations associated with a particular rule:

- Keep columns with step number 6
- Delete column with step number 5
- Rename column with step number 4
- Add column with step number 3
- Rename table with step number 2
- Rename schema with step number 1

With this ordering specified, rename schema transformations are performed first, and delete column transformations are performed last.

Considerations for Rule-Based Transformations

Several considerations apply to declarative rule-based transformations.

These considerations include the following:

- For a rule-based transformation to be performed by an XStream component, the rule must be in the positive rule set for the XStream component. If the rule is in the negative rule set for the XStream component, then the XStream component ignores the rule-based transformation.
- Rule-based transformations are different from transformations performed using the `DBMS_TRANSFORM` package. This document does not discuss transformations performed with the `DBMS_TRANSFORM` package.
- If a large percentage of row LCRs will be transformed in an XStream In configuration, you can use DML handlers with XStream In. Be aware that this method may not perform as well as making the changes in the XStream In client application. If you are performing multiple or complex transformations on row LCRs in an XStream In configuration, then consider reducing the XStream In processing time by making these modifications in the client application prior to sending the changes to XStream In.

XStream and the Oracle Replication Performance Advisor

The Oracle Replication Performance Advisor consists a collection of data dictionary views.

The 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 Performance Advisor also provides information about how Oracle Replication components are performing.

Apply processes function as XStream outbound servers and inbound servers. In general, the Performance Advisor works the same way for an Oracle Replication environment with apply processes and an XStream environment with outbound servers or inbound servers. This section describes important considerations about using the Performance Advisor in an XStream environment.

- XStream Components
  The Performance Advisor tracks several XStream components.
- Topology and Stream Paths
  In the Oracle Replication topology, a stream path is a flow of LCRs from a source to a destination.
- XStream and Component-Level Statistics
  The Performance Advisor tracks component-level statistics.
- The UTL_RPADV Package
  The `UTL_RPADV` package automates the collection of statistics associated with XStream performance.

XStream Components

The Performance Advisor tracks several XStream components.

The 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 Replication environment and an XStream environment, except for `APPLY`. The `APPLY` component type can be an XStream outbound server or inbound server.

In addition, the 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.

- XStream Out Apply Subcomponents
  There are several XStream Out apply subcomponents types.
- XStream In Apply Subcomponents
  There are several XStream In apply subcomponents types.

XStream Out Apply Subcomponents

There are several XStream Out apply subcomponents types.

The following subcomponent types are possible:

- `PROPAGATION SENDER+RECEIVER` for sending LCRs from a capture process to an outbound server where the capture process and outbound server are in different databases.
- `APPLY READER` for a reader server. `APPLY READER` receives LCRs from the capture process, organizes them into transactions, does dependency calculations, and passes the LCRs to the apply coordinator.
- `APPLY COORDINATOR` for a coordinator process. It takes the transactions from the capture process, uses the dependency information to determine how to schedule the transactions and sends the LCRs to the apply server.
- `APPLY SERVER` for an apply server. It delivers the LCRs to the client application.

XStream In Apply Subcomponents

There are several XStream In apply subcomponents types.

The following subcomponent types are possible:

- `APPLY READER` for a reader server. It takes the LCRs from client application converts them into transactions, checks the transactional order and does dependency calculations.
- `APPLY COORDINATOR` for a coordinator process. It takes the transactions from the reader server, uses the dependency information to determine how to schedule the transactions and sends the LCRs to the apply server.
- `APPLY SERVER` for an apply server. It applies the LCRs to an apply handler. If the LCR cannot be applied, it is placed into an error queue.

Topology and Stream Paths

In the Oracle Replication topology, a stream path is a flow of LCRs from a source to a destination.

A stream path begins with a capture process or XStream In client application. A stream path ends where an apply process, outbound server, or inbound server receives the LCRs. The stream path might flow through multiple source and destination components 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 Replication topology only gathers information about a stream path if the stream path ends with an apply process, an outbound server, or an inbound server.

XStream and Component-Level Statistics

The Performance Advisor tracks component-level statistics.

The Performance Advisor tracks the following component-level statistics:

- The `MESSAGE APPLY RATE` is the average number of LCRs 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 LCRs.

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 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 LCR was created at a source database and when the LCR was applied by the apply process at the destination database.
- For outbound servers, the apply `LATENCY` is amount of time between when the LCR was created at a source database and when the LCR was sent to the XStream client application.
- For inbound servers, the apply `LATENCY` is amount of time between when the LCR was created by the XStream client application and when the LCR 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.

The UTL_RPADV Package

The `UTL_RPADV` package automates the collection of statistics associated with XStream performance.

`UTL_RPADV` provides a series of subprograms that collect and analyze statistics for the XStream components in a distributed database environment. The package uses the Performance Advisor and the `COLLECT_STATS` procedure to automate the collection of statistics.

The output is formatted so that it can be imported into a spreadsheet easily and analyzed. You can examine XStream performance statistics output with the `UTL_RPADV.SHOW_STATS` procedure or view the same information in an HTML-formatted report with the `UTL_RPADV.SHOW_STATS_HTML` procedure.

The `UTL_RPADV` package works essentially the same way for an Oracle Replication environment with apply processes as it does for an XStream environment with outbound servers or inbound servers. Since XStream is concerned with data changes to or from a client application, the output of the `SHOW_STATS` procedure is different for XStream than for Oracle Replication.

- Collecting XStream Statistics Using the UTL_RPADV Package
  You can collect XStream statistics with the `UTL_RPADV` package.
- Showing XStream Statistics on the Command Line
  The `SHOW_STATS` procedure in the `UTL_RPADV` package displays the statistics that the Performance Advisor gathered and stored.
- Interpreting SHOW_STATS Output
  There are differences between the output for apply processes and the output for XStream outbound servers and inbound servers.

- Showing XStream Statistics in an HTML Report
  The `SHOW_STATS_HTML` procedure in the `UTL_RPADV` package creates an HTML report that contains the statistics that the Performance Advisor gathered and stored.
- Interpreting the HTML Report From SHOW_STATS_HTML
  The `SHOW_STATS_HTML` procedure in the `UTL_RPADV` package can generate the same output as the `SHOW_STATS` procedure, but it formats the output as HTML in HTML files.

Collecting XStream Statistics Using the UTL_RPADV Package

You can collect XStream statistics with the `UTL_RPADV` package.

To collect XStream statistics using the `UTL_RPADV` package, complete the following steps:

1. Identify the database that you will use to gather the information. An administrative user at this database must meet the following requirements:
   - If you want to gather XStream statistics for more than one database, the user must have access to a database link to each database that contains XStream components to monitor.
   - The user must have been granted privileges using the `XSTREAM_CAPTURE or XSTREAM_APPLY` roles. If you want to gather XStream statistics for more than one database, each database link must connect to a user at the remote database that has been granted the `XSTREAM_CAPTURE or XSTREAM_APPLY` roles.

     If you configure an XStream administrator at each database with XStream components, then the XStream administrator has the necessary privileges.
   - The user must have the necessary privileges to create tables and procedures. If you want to gather XStream statistics for more than one database, each database link must connect to a user at the remote database that has the necessary privileges to create tables and procedures.

   If no database in your environment meets these requirements, then choose a database, configure the necessary database links, and grant the necessary privileges to the users before proceeding.
2. In SQL*Plus, connect to the database you identified in Step 1 as a user that meets the requirements listed in Step 1.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.
3. Run the `utlrpadv.sql` script in the rdbms/admin directory in `ORACLE_HOME` to load the `UTL_RPADV` package. For example:

   ```
   @utlrpadv.sql
   ```
4. Either collect the current XStream performance statistics once, or create a job that continually monitors XStream performance:
   - To collect the current XStream performance statistics once, run the `COLLECT_STATS` procedure:

     ```
     exec UTL_RPADV.COLLECT_STATS
     ```

     This example uses the default values for the parameters in the `COLLECT_STATS` procedure. Therefore, this example runs the Performance Advisor 10 times with 60 seconds between each run. These values correspond with the default values for the `num_runs` and `interval` parameters, respectively, in the `COLLECT_STATS` procedure.
   - To create a job that continually monitors XStream performance:

```
exec UTL_RPADV.START_MONITORING
```

This example creates a monitoring job, and the monitoring job gathers performance statistics continually at set intervals. This example uses the default values for the parameters in the START_MONITORING procedure. Therefore, this example runs the Performance Advisor every 60 seconds. This value corresponds with the default value for the interval parameter in the START_MONITORING procedure. If an interval is specified in the START_MONITORING procedure, then the specified interval is used for the interval parameter in the COLLECT_STATS procedure.

These procedures include several parameters that you can use to adjust the way performance statistics are gathered. See *Oracle Database PL/SQL Packages and Types Reference* for more information.

You can show the statistics by running the SHOW_STATS procedure. See "Showing XStream Statistics on the Command Line".

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the UTL_RPADV package

Showing XStream Statistics on the Command Line

The SHOW_STATS procedure in the UTL_RPADV package displays the statistics that the Performance Advisor gathered and stored.

Use the path_stat_table parameter to specify the table that contains the statistics.

When you gather statistics using the COLLECT_STATS procedure, this table is specified in the path_stat_table parameter in the COLLECT_STATS procedure. By default, the table name is STREAMS$_ADVISOR_PATH_STAT.

When you gather statistics using the START_MONITORING procedure, you can determine the name for this table by querying the SHOW_STATS_TABLE column in the STREAMS$_PA_MONITORING view. The default table for a monitoring job is STREAMS$_PA_SHOW_PATH_STAT.

To show statistics collected using the UTL_RPADV package and stored in the STREAMS$_ADVISOR_PATH_STAT table, complete the following steps:

1. Collect statistics by completing the steps described in "Collecting XStream Statistics Using the UTL_RPADV Package".
2. Connect to the database as the user who collected the statistics.
3. If you are using a monitoring job, then query the SHOW_STATS_TABLE column in the STREAMS$_PA_MONITORING view to determine the name of this table that stores the statistics:

   ```
   SELECT SHOW_STATS_TABLE FROM STREAMS$_PA_MONITORING;
   ```
4. Run the SHOW_STATS procedure.

   For example, if you are using a monitoring job and the default storage table, then run the following procedure:

```
SET SERVEROUTPUT ON SIZE 50000
BEGIN
  UTL_RPADV.SHOW_STATS(
    path_stat_table => 'STREAMS$_PA_SHOW_PATH_STAT');
END;
/
```

Interpreting SHOW_STATS Output

There are 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_RPADV` package and the `SHOW_STATS` output for apply processes.

- Sample Output When an Outbound Server Is the Last Component in a Path
  Here is sample output for when an outbound server is the last component in a path.
- Sample Output When an Inbound Server Is the Last Component in a Path
  Here is sample output for when an inbound server is the last component in a path.

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about using the `UTL_RPADV` package

Sample Output When an Outbound Server Is the Last Component in a Path

Here is sample output for when an outbound server is the last component in a path.

```
LEGEND
<statistics>=<capture>  [<queue>  <psender>  <preceiver>  <queue>  ]<apply>
<bottleneck>
<capture>    = '|<C>'<name>  <msgs captured/sec>  <msgs enqueued/sec>  <latency>
    'LMR'<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'LMP' (<parallelism>)<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'LMB'<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'CAP'<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'CAP+PS'<msgs sent/sec>  <bytes sent/sec>  <latency>  <idl%>
<flwctrl%>  <topevt%>  <topevt>
<apply>      = '|<A>'<name>  <msgs applied/sec>  <txns applied/sec>  <latency>
    'PS+PR'<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'APR'<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'APC'<idl%>  <flwctrl%>  <topevt%>  <topevt>
    'APS' (<parallelism>)<idl%>  <flwctrl%>  <topevt%>  <topevt>
<queue>      = '|<Q>'<name>  <msgs enqueued/sec>  <msgs spilled/sec>  <msgs in
queue>
<psender>    = '|<PS>'<name>  <msgs sent/sec>  <bytes sent/sec>  <latency>  <idl%>
<flwctrl%>  <topevt%>  <topevt>
<preceiver>  = '|<PR>'<name>  <idl%>  <flwctrl%>  <topevt%>  <topevt>
<bottleneck>= '|<B>'<name>  <sub_name>  <sessionid>  <serial#>  <topevt%>  <topevt>
```

```
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 the "SQL*Net more data to client" performance metric for an apply server is measured like 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

Here 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. If you and the abbreviations to determine the statistics in the output.

Showing XStream Statistics in an HTML Report

The `SHOW_STATS_HTML` procedure in the `UTL_RPADV` package creates an HTML report that contains the statistics that the Performance Advisor gathered and stored.

Use the `comp_stat_table` parameter to specify the table that contains the statistics.

When you gather statistics using the `COLLECT_STATS` procedure, this table is specified in the `comp_stat_table` parameter in the `COLLECT_STATS` procedure. By default, the table name is `STREAMS$_ADVISOR_COMP_STAT`.

When you gather statistics using the `START_MONITORING` procedure, you can determine the name for this table by querying the `SHOW_STATS_TABLE` column in the `STREAMS$_PA_MONITORING` view. The default table for a monitoring job is `STREAMS$_ADVISOR_COMP_STAT`.

The default for the `comp_stat_table` parameter is `STREAMS$_ADVISOR_COMP_STAT`. Ensure that you specify the correct table when you run the `SHOW_STATS_HTML` procedure.

The `SHOW_STATS_HTML` procedure must store the HTML report in a directory. Use the `directory` parameter to specify a directory object.

To show statistics collected using the `UTL_RPADV` package and stored in the `STREAMS$_ADVISOR_COMP_STAT` table, complete the following steps:

1. Collect statistics by completing the steps described in "Collecting XStream Statistics Using the UTL_RPADV Package".
2. Connect to the database as the user who collected the statistics.
3. If you are using a monitoring job, then query the `SHOW_STATS_TABLE` column in the `STREAMS$_PA_MONITORING` view to determine the name of this table that stores the statistics:

   ```
   SELECT SHOW_STATS_TABLE FROM STREAMS$_PA_MONITORING;
   ```
4. Create a directory object for the directory that will contain the files in the HTML report.

   For example, to create a directory object named `XSTREAM_STATS_HTML` for the /usr/xstream/reports directory, run the following SQL statement:

   ```
   CREATE DIRECTORY XSTREAM_STATS_HTML AS '/usr/xstream/reports';
   ```
5. Run the `SHOW_STATS_HTML` procedure.

For example, if you are using a monitoring job and the default storage table, then run the following procedure:

```
BEGIN
  UTL_RPADV.SHOW_STATS_HTML(
    directory      => 'XSTREAM_STATS_HTML',
    reportname     => 'xstream_stats.html',
    comp_stat_table => 'STREAMS$_ADVISOR_COMP_STAT');
END;
/
```

Interpreting the HTML Report From SHOW_STATS_HTML

The `SHOW_STATS_HTML` procedure in the `UTL_RPADV` package can generate the same output as the `SHOW_STATS` procedure, but it formats the output as HTML in HTML files.

The `SHOW_STATS_HTML` output is easier to read than the `SHOW_STATS` output. For example, the procedure generates multiple files, and each file begins with the report name. The report includes tables with the performance statistics. Statistics for different paths are in different rows in these tables, and you can click on a path for more detailed statistics about the path. The `report_name` parameter indicates the master HTML file with links to the other HTML files.

The following are considerations for using the `SHOW_STATS_HTML` procedure:

- The default table that stores the statistics is `STREAMS$_ADVISOR_COMP_STAT`. The `SHOW_STATS` procedure uses a different default table (`STREAMS$_ADVISOR_PATH_STAT`).
- You must specify a directory object in the `directory` parameter of the procedure. This directory stores the HTML files generated by the procedure.

  The specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes the procedure must have `READ` and `WRITE` privilege on the directory.

> **See Also:**
>
> "Interpreting SHOW_STATS Output"

Using Automatic Workload Repository (AWR) Reports for Oracle Database

Automatic Workload Repository (AWR) is a good starting point for identifying general database performance issues, which can provide indicators to help locate problems with XStream Out or XStream In processes. Using AWR, you can easily determine if the bottlenecks are inside or outside of the database.

Starting with Oracle Database 23ai, AWR queries and reports are simplified and enhanced to present the data in an easy to understand view of the XStream In process. With these report views, it would be easier to troubleshoot XStream In performance issues by categorizing them as one of the following:

- Workload issue
- Misconfiguration issue, such as slow XStream In SQL due to lack of indexes

- Performance bottleneck issue at the database side or in the Oracle Database XStream processes outside of the database

The XStream Inion sections of the enhanced AWR reports include the following features:

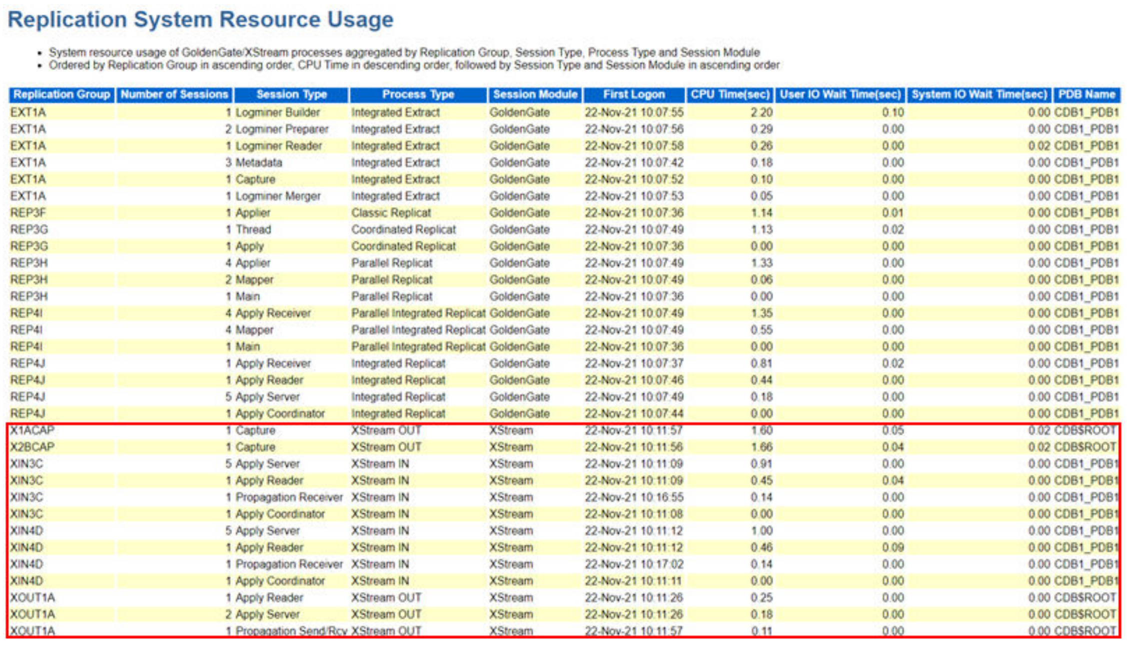
- A more complete XStream Inion System Resource Usage section, which shows the system resource usage for all Oracle GoldenGate and XStream XStream In processes, whether they be foreground or background, and presented on a per PDB XStream Out or XStream In basis.
- A separate section for XStream Inion related Top SQL statistics to make it easier to identify XStream Inion related SQL performance issues. The number of Top SQL shown is a fraction of the Top SQL shown for the database.
- A separate section for top wait events for XStream In related processes to easily troubleshoot XStream Inion related performance problems.
- Reorganized XStream Inion related sections to present the XStream In statistics organized by individual XStream Out, XStream In, and XStream type.
- Customized information for XStream In or XStream Out processes.

Topics:

- Replication System Resource Usage
- Replication Top SQLs

Replication System Resource Usage

Oracle Database XStream replication process name, process type and number of sessions of its sub-components are displayed in the metrics. The performance statistic are aggregated by the functionality of process sub-components, group by process name. This allows the ability to monitor resource usage at per XStream Out and XStream In, with detailed information of all its sub-components.



Replication System Resource Usage

- System resource usage of GoldenGate/XStream processes aggregated by Replication Group, Session Type, Process Type and Session Module
- Ordered by Replication Group in ascending order, CPU Time in descending order, followed by Session Type and Session Module in ascending order

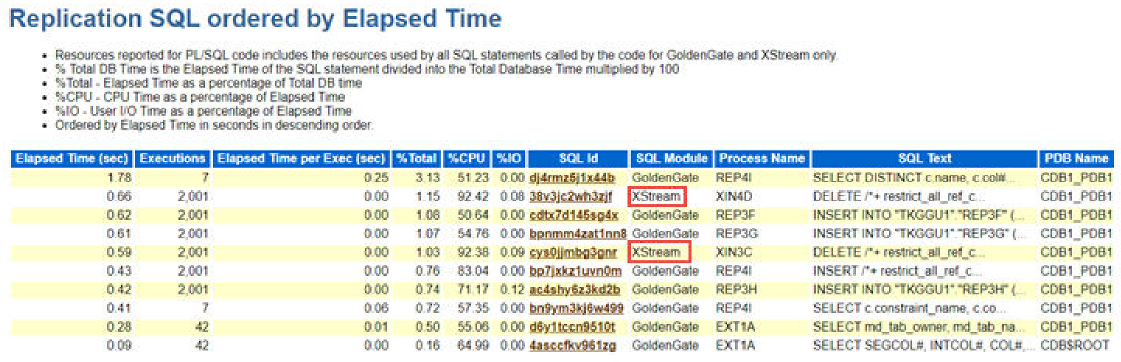
Replication Group	Number of Sessions	Session Type	Process Type	Session Module	First Logon	CPU Time(sec)	User IO Wait Time(sec)	System IO Wait Time(sec)	PDB Name
EXT1A	1	Logminer Builder	Integrated Extract	GoldenGate	22-Nov-21 10:07:55	2.20	0.10	0.00	CDB1_PDB1
EXT1A	2	Logminer Preparer	Integrated Extract	GoldenGate	22-Nov-21 10:07:56	0.29	0.00	0.00	CDB1_PDB1
EXT1A	1	Logminer Reader	Integrated Extract	GoldenGate	22-Nov-21 10:07:58	0.26	0.00	0.02	CDB1_PDB1
EXT1A	3	Metadata	Integrated Extract	GoldenGate	22-Nov-21 10:07:42	0.18	0.00	0.00	CDB1_PDB1
EXT1A	1	Capture	Integrated Extract	GoldenGate	22-Nov-21 10:07:52	0.10	0.00	0.00	CDB1_PDB1
EXT1A	1	Logminer Merger	Integrated Extract	GoldenGate	22-Nov-21 10:07:53	0.05	0.00	0.00	CDB1_PDB1
REP3F	1	Applier	Classic Replicat	GoldenGate	22-Nov-21 10:07:36	1.14	0.01	0.00	CDB1_PDB1
REP3G	1	Thread	Coordinated Replicat	GoldenGate	22-Nov-21 10:07:49	1.13	0.02	0.00	CDB1_PDB1
REP3G	1	Apply	Coordinated Replicat	GoldenGate	22-Nov-21 10:07:36	0.00	0.00	0.00	CDB1_PDB1
REP3H	4	Applier	Parallel Replicat	GoldenGate	22-Nov-21 10:07:49	1.33	0.00	0.00	CDB1_PDB1
REP3H	2	Mapper	Parallel Replicat	GoldenGate	22-Nov-21 10:07:49	0.06	0.00	0.00	CDB1_PDB1
REP3H	1	Main	Parallel Replicat	GoldenGate	22-Nov-21 10:07:36	0.00	0.00	0.00	CDB1_PDB1
REP4I	4	Apply Receiver	Parallel Integrated Replicat	GoldenGate	22-Nov-21 10:07:49	1.35	0.00	0.00	CDB1_PDB1
REP4I	4	Mapper	Parallel Integrated Replicat	GoldenGate	22-Nov-21 10:07:49	0.55	0.00	0.00	CDB1_PDB1
REP4I	1	Main	Parallel Integrated Replicat	GoldenGate	22-Nov-21 10:07:36	0.00	0.00	0.00	CDB1_PDB1
REP4J	1	Apply Receiver	Integrated Replicat	GoldenGate	22-Nov-21 10:07:37	0.81	0.02	0.00	CDB1_PDB1
REP4J	1	Apply Reader	Integrated Replicat	GoldenGate	22-Nov-21 10:07:46	0.44	0.00	0.00	CDB1_PDB1
REP4J	5	Apply Server	Integrated Replicat	GoldenGate	22-Nov-21 10:07:49	0.18	0.00	0.00	CDB1_PDB1
REP4J	1	Apply Coordinator	Integrated Replicat	GoldenGate	22-Nov-21 10:07:44	0.00	0.00	0.00	CDB1_PDB1
X1ACAP	1	Capture	XStream OUT	XStream	22-Nov-21 10:11:57	1.60	0.05	0.02	CDB$ROOT
X2BCAP	1	Capture	XStream OUT	XStream	22-Nov-21 10:11:56	1.66	0.04	0.02	CDB$ROOT
XIN3C	5	Apply Server	XStream IN	XStream	22-Nov-21 10:11:09	0.91	0.00	0.00	CDB1_PDB1
XIN3C	1	Apply Reader	XStream IN	XStream	22-Nov-21 10:11:09	0.45	0.04	0.00	CDB1_PDB1
XIN3C	1	Propagation Receiver	XStream IN	XStream	22-Nov-21 10:16:55	0.14	0.00	0.00	CDB1_PDB1
XIN3C	1	Apply Coordinator	XStream IN	XStream	22-Nov-21 10:11:08	0.00	0.00	0.00	CDB1_PDB1
XIN4D	5	Apply Server	XStream IN	XStream	22-Nov-21 10:11:12	1.00	0.00	0.00	CDB1_PDB1
XIN4D	1	Apply Reader	XStream IN	XStream	22-Nov-21 10:11:12	0.46	0.09	0.00	CDB1_PDB1
XIN4D	1	Propagation Receiver	XStream IN	XStream	22-Nov-21 10:17:02	0.14	0.00	0.00	CDB1_PDB1
XIN4D	1	Apply Coordinator	XStream IN	XStream	22-Nov-21 10:11:11	0.00	0.00	0.00	CDB1_PDB1
XOUT1A	1	Apply Reader	XStream OUT	XStream	22-Nov-21 10:11:26	0.25	0.00	0.00	CDB$ROOT
XOUT1A	2	Apply Server	XStream OUT	XStream	22-Nov-21 10:11:26	0.18	0.00	0.00	CDB$ROOT
XOUT1A	1	Propagation Send/Rcv	XStream OUT	XStream	22-Nov-21 10:11:57	0.11	0.00	0.00	CDB$ROOT

Replication Top SQLs

SQLs executed by replication processes, displayed in different sections ordered by Elapsed Time, CPU Time and Execution are presented. Replication process name is added for identifying the process that is executing the SQL.

While the common TOP SQL section of AWR is related to all SQL statements within the database, the Top SQL within the Replication section is focused on XStream and Oracle GoldenGate related SQL only. Therefore, more replication information is visible in this section.

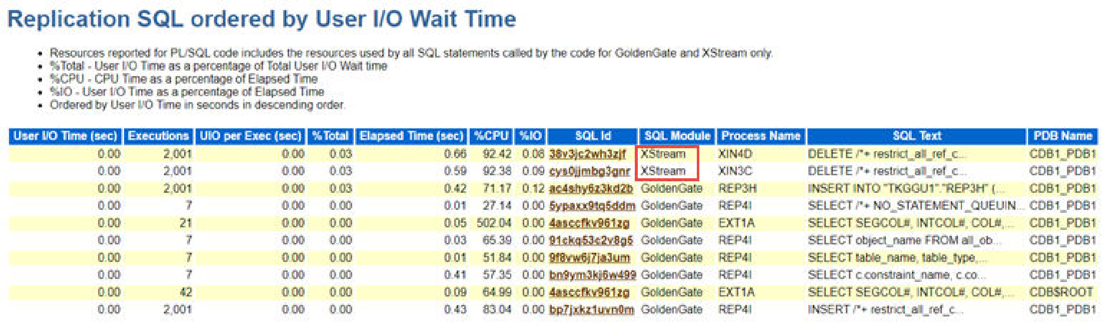
Here are examples of the top SQL reports.



Replication SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code for GoldenGate and XStream only.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Ordered by Elapsed Time in seconds in descending order.

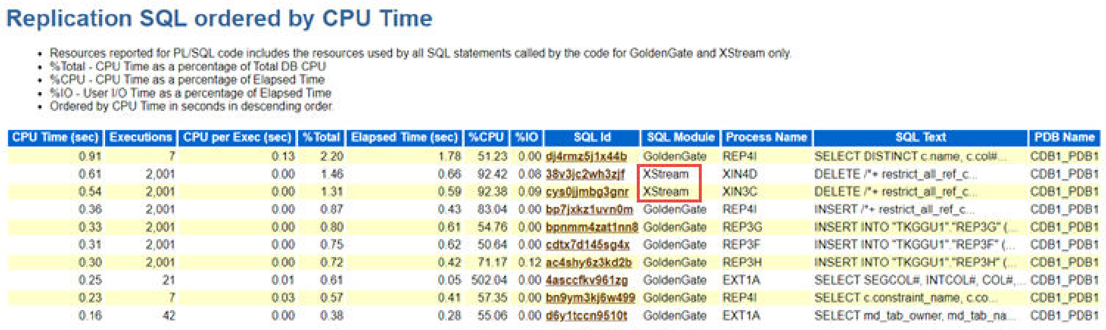
Elapsed Time (sec)	Executions	Elapsed Time per Exec (sec)	%Total	%CPU	%IO	SQL Id	SQL Module	Process Name	SQL Text	PDB Name
1.78	7	0.25	3.13	51.23	0.00	dj4rmz5j1x44b	GoldenGate	REP4I	SELECT DISTINCT c.name, c.col#...	CDB1_PDB1
0.66	2,001	0.00	1.15	92.42	0.08	38v3jc2wh3zjf	XStream	XIN4D	DELETE /*+ restrict_all_ref_c...	CDB1_PDB1
0.62	2,001	0.00	1.08	50.64	0.00	cdtx7d145sg4x	GoldenGate	REP3F	INSERT INTO "TKGGU1"."REP3F" (...	CDB1_PDB1
0.61	2,001	0.00	1.07	54.76	0.00	bpnmm4zat1nn8	GoldenGate	REP3G	INSERT INTO "TKGGU1"."REP3G" (...	CDB1_PDB1
0.59	2,001	0.00	1.03	92.38	0.09	cys0jjmbg3gnr	XStream	XIN3C	DELETE /*+ restrict_all_ref_c...	CDB1_PDB1
0.43	2,001	0.00	0.76	83.04	0.00	bp7jxkz1uvn0m	GoldenGate	REP4I	INSERT /*+ restrict_all_ref_c...	CDB1_PDB1
0.42	2,001	0.00	0.74	71.17	0.12	ac4shy6z3kd2b	GoldenGate	REP3H	INSERT INTO "TKGGU1"."REP3H" (...	CDB1_PDB1
0.41	7	0.06	0.72	57.35	0.00	bn9ym3kj6w499	GoldenGate	REP4I	SELECT c.constraint_name, c.co...	CDB1_PDB1
0.28	42	0.01	0.50	55.06	0.00	d6y1tccn9510t	GoldenGate	EXT1A	SELECT md_tab_owner, md_tab_na...	CDB1_PDB1
0.09	42	0.00	0.16	64.99	0.00	4asccfkv961zg	GoldenGate	EXT1A	SELECT SEGCOL#, INTCOL#, COL#,...	CDB$ROOT



Replication SQL ordered by User I/O Wait Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code for GoldenGate and XStream only.
- %Total - User I/O Time as a percentage of Total User I/O Wait time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Ordered by User I/O Time in seconds in descending order.

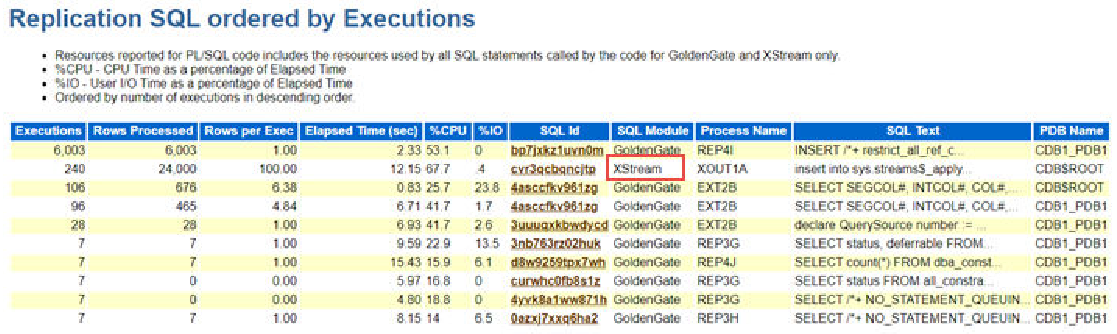
User I/O Time (sec)	Executions	UIO per Exec (sec)	%Total	Elapsed Time (sec)	%CPU	%IO	SQL Id	SQL Module	Process Name	SQL Text	PDB Name
0.00	2,001	0.00	0.03	0.66	92.42	0.08	38v3jc2wh3zjf	XStream	XIN4D	DELETE /*+ restrict_all_ref_c...	CDB1_PDB1
0.00	2,001	0.00	0.03	0.59	92.38	0.09	cys0jjmbg3gnr	XStream	XIN3C	DELETE /*+ restrict_all_ref_c...	CDB1_PDB1
0.00	2,001	0.00	0.03	0.42	71.17	0.12	ac4shy6z3kd2b	GoldenGate	REP3H	INSERT INTO "TKGGU1"."REP3H" (...	CDB1_PDB1
0.00	7	0.00	0.00	0.01	27.14	0.00	5ypaxx9tq5ddm	GoldenGate	REP4I	SELECT /*+ NO_STATEMENT_QUEUIN...	CDB1_PDB1
0.00	21	0.00	0.00	0.05	502.04	0.00	4asccfkv961zg	GoldenGate	EXT1A	SELECT SEGCOL#, INTCOL#, COL#,...	CDB1_PDB1
0.00	7	0.00	0.00	0.03	65.39	0.00	91ckq53c2v8g5	GoldenGate	REP4I	SELECT object_name FROM all_ob...	CDB1_PDB1
0.00	7	0.00	0.00	0.01	51.84	0.00	9f8vw6j7ja3um	GoldenGate	REP4I	SELECT table_name, table_type,...	CDB1_PDB1
0.00	7	0.00	0.00	0.41	57.35	0.00	bn9ym3kj6w499	GoldenGate	REP4I	SELECT c.constraint_name, c.co...	CDB1_PDB1
0.00	42	0.00	0.00	0.09	64.99	0.00	4asccfkv961zg	GoldenGate	EXT1A	SELECT SEGCOL#, INTCOL#, COL#,...	CDB$ROOT
0.00	2,001	0.00	0.00	0.43	83.04	0.00	bp7jxkz1uvn0m	GoldenGate	REP4I	INSERT /*+ restrict_all_ref_c...	CDB1_PDB1



Replication SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code for GoldenGate and XStream only.
- %Total - CPU Time as a percentage of Total DB CPU
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Ordered by CPU Time in seconds in descending order.

CPU Time (sec)	Executions	CPU per Exec (sec)	%Total	Elapsed Time (sec)	%CPU	%IO	SQL Id	SQL Module	Process Name	SQL Text	PDB Name
0.91	7	0.13	2.20	1.78	51.23	0.00	dj4rmz5j1x44b	GoldenGate	REP4I	SELECT DISTINCT c.name, c.col#...	CDB1_PDB1
0.61	2,001	0.00	1.46	0.66	92.42	0.08	38v3jc2wh3zjf	XStream	XIN4D	DELETE /*+ restrict_all_ref_c...	CDB1_PDB1
0.54	2,001	0.00	1.31	0.59	92.38	0.09	cys0jjmbg3gnr	XStream	XIN3C	DELETE /*+ restrict_all_ref_c...	CDB1_PDB1
0.36	2,001	0.00	0.87	0.43	83.04	0.00	bp7jxkz1uvn0m	GoldenGate	REP4I	INSERT /*+ restrict_all_ref_c...	CDB1_PDB1
0.33	2,001	0.00	0.80	0.61	54.76	0.00	bpnmm4zat1nn8	GoldenGate	REP3G	INSERT INTO "TKGGU1"."REP3G" (...	CDB1_PDB1
0.31	2,001	0.00	0.75	0.62	50.64	0.00	cdtx7d145sg4x	GoldenGate	REP3F	INSERT INTO "TKGGU1"."REP3F" (...	CDB1_PDB1
0.30	2,001	0.00	0.72	0.42	71.17	0.12	ac4shy6z3kd2b	GoldenGate	REP3H	INSERT INTO "TKGGU1"."REP3H" (...	CDB1_PDB1
0.25	21	0.01	0.61	0.05	502.04	0.00	4asccfkv961zg	GoldenGate	EXT1A	SELECT SEGCOL#, INTCOL#, COL#,...	CDB1_PDB1
0.23	7	0.03	0.57	0.41	57.35	0.00	bn9ym3kj6w499	GoldenGate	REP4I	SELECT c.constraint_name, c.co...	CDB1_PDB1
0.16	42	0.00	0.38	0.28	55.06	0.00	d6y1tccn9510t	GoldenGate	EXT1A	SELECT md_tab_owner, md_tab_na...	CDB1_PDB1



Replication SQL ordered by Executions

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code for GoldenGate and XStream only.
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Ordered by number of executions in descending order.

Executions	Rows Processed	Rows per Exec	Elapsed Time (sec)	%CPU	%IO	SQL Id	SQL Module	Process Name	SQL Text	PDB Name
6,003	6,003	1.00	2.33	53.1	0	bp7jxkz1uvn0m	GoldenGate	REP4I	INSERT /*+ restrict_all_ref_c...	CDB1_PDB1
240	24,000	100.00	12.15	67.7	.4	cvr3qcbqncjtp	XStream	XOUT1A	insert into sys.streams$_apply...	CDB$ROOT
106	676	6.38	0.83	25.7	23.8	4asccfkv961zg	GoldenGate	EXT2B	SELECT SEGCOL#, INTCOL#, COL#,...	CDB$ROOT
96	465	4.84	6.71	41.7	1.7	4asccfkv961zg	GoldenGate	EXT2B	SELECT SEGCOL#, INTCOL#, COL#,...	CDB1_PDB1
28	28	1.00	6.93	41.7	2.6	3uuuqxkbwdycd	GoldenGate	EXT2B	declare QuerySource number := ...	CDB1_PDB1
7	7	1.00	9.59	22.9	13.5	3nb763rz02huk	GoldenGate	REP3G	SELECT status, deferrable FROM...	CDB1_PDB1
7	7	1.00	15.43	15.9	6.1	d8w9259tpx7wh	GoldenGate	REP4J	SELECT count(*) FROM dba_const...	CDB1_PDB1
7	0	0.00	5.97	16.8	0	curwhc0fb8s1z	GoldenGate	REP3G	SELECT status FROM all_constra...	CDB1_PDB1
7	0	0.00	4.80	18.8	0	4yvk8a1ww871h	GoldenGate	REP3G	SELECT /*+ NO_STATEMENT_QUEUIN...	CDB1_PDB1
7	7	1.00	8.15	14	6.5	0azxj7xxq6ha2	GoldenGate	REP3H	SELECT /*+ NO_STATEMENT_QUEUIN...	CDB1_PDB1

Automatic Workload Repository (AWR) Report for XStream

Automatic Workload Repository (AWR) reports help leverage the existing statistics from AWR tables.

The AWR report provides information about system resource usage, top SQL statistics and wait events, and helps organize replication process performance.

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

XStream In

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

XStream In consists of the following sub-components:

- Apply Network receiver
- Apply Reader
- Apply Coordinator
- Apply Server

Table 2-10 XStream In

XStream In Name	XStream In Client Name
Apply Name	Apply process name
LCRs Buffered (Reader)	Number of LCRs buffered waiting to be processed by Apply Reader
Txns Received (Coordinator)	Number of Txns received by Apply Coordinator
Txns Not Assigned (Coordinator)	Total number of Txns not assigned yet by Coordinator
Txns Assigned (Apply Server)	Number of Txns assigned to Apply Server
LCRs Applied (Apply Server)	Number of LCRs applied by Apply Server
Txns with Dependencies %	Percentage of transactions having to wait for other transactions due to dependency

Table 2-10 (Cont.) XStream In

XStream In Name	XStream In Client Name
Txns with Watermark Dependency %	Percentage of transactions having to wait due to source transaction commit ordering
Total Errors	Total Txns with errors

XStream Out

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

XStream Out configuration consists of a capture engine that captures Data Manipulation Language (DML) and DML changes, and an Outbound Server sends those changes in the logical change record (LCR) format to the client.

XStream Out consists of the following sub-components:

- Logminer Engine (Reader, Builder, Preparer, Merger when parallelism > 0)
- Capture Engine
- Outbound Server:
  - Apply Network Receiver (ANR)
  - Apply Coordinator
  - Apply Server

Table 2-11 XStream Out Capture

XStream Out Name	XStream Out Client Application Name
XStream Out Name	XStream Out client application name
Capture Name	Capture process name
Begin Lag(s)	Time when the most recent LCR received – creation time of the most recent LCR at the start of the snapshot interval
End Lag(s)	Time when the most recent LCR received – creation time of the most recent LCR at the end of the snapshot interval
Redo Mined(B)	The amount of redo data mined (in bytes) since the Capture process last started within the snapshot interval
Redo Mined(B) per Sec	Redo Mined Rate derived from the snapshot interval
Bytes Sent	Number of bytes sent by the Capture process to the Extract process since the last time the Extract process attached to the Capture process
Capture LCRs	Number of LCRs delivered to capture from Logminer
Capture LCRs per Sec	Capture LCRs Rate
Sent LCRs per Sec	Sent LCRs Rate
Redo Wait Time(sec)	Time spent by the Capture process in the `WAITING FOR REDO` state
Redo Wait Time per LCR	Redo Wait Time per LCR
Pause Time(sec)	Time the Capture spent in flow control, waiting for downstream Outbound Server
Pause Time (s)/LCR	Pause Time per LCR

Table 2-12 XStream Out Outbound Server

XStream Out Name	XStream Out Client Name
Outbound Server	Outbound server name
LCRs Buffered	Number of LCRs buffered
Txns Received	Number of Txns received by Apply Coordinator
Txns Available to Send	Number of Txns available to send to Apply Server
LCRs per Sec Sent	LCR sent rate

XStream and SQL Generation

SQL generation is the ability to generate the SQL statement required to perform the change encapsulated in a row LCR.

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.

- Interfaces for Performing SQL Generation
  You can use different interfaces for SQL generation.
- SQL Generation Formats
  SQL statements can be generated in one of two formats: inline values or bind variables.
- SQL Generation and Data Types
  SQL generation supports most data types.
- SQL Generation and Character Sets
  When you use the LCR methods, the generated SQL is in the database character set.
- Sample Generated SQL Statements
  Examples illustrate generated SQL statements.
- SQL Generation Demo
  A demo that performs SQL generation is available.

Interfaces for Performing SQL Generation

You can use different interfaces for 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
- *Oracle Call Interface Developer's Guide*
- *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*
- *Oracle Call Interface Developer's Guide*
- 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 Also:

Oracle Database PL/SQL Language Reference for more information about SQL injection

SQL Generation and Data Types

SQL generation supports most data types.

SQL generation supports the following data types:

- `VARCHAR2`

- `NVARCHAR2`
- `NUMBER`
- `FLOAT`
- `DATE`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `LONG`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `LONG RAW`
- `CHAR`
- `NCHAR`
- `CLOB` with `BASICFILE` storage
- `NCLOB` with `BASICFILE` storage
- `BLOB` with `BASICFILE` storage
- `XMLType` stored as `CLOB`, object relational, or as binary XML
- `JSON`
- `BOOLEAN`

Note:

- The maximum size of the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types has been increased in Oracle Database 12*c* when the `COMPATIBLE` initialization parameter is set to `12.0.0` and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`.
- `XMLType` stored as a `CLOB` is deprecated in Oracle Database 12*c* Release 1 (12.1).

- SQL Generation and Automatic Data Type Conversion
  An XStream outbound server or inbound server performs implicit data type conversion where it is possible, and the generated SQL follows ANSI standards where it is possible.
- SQL Generation and LOB, LONG, LONG RAW, and XMLType Data Types
  For `INSERT` and `UPDATE` operations on LOB columns, an outbound server automatically assembles the LOB chunks using LOB assembly.

See Also:

Oracle Database SQL Language Reference for information about data types

SQL Generation and Automatic Data Type Conversion

An XStream outbound server or inbound server performs implicit data type conversion where it is possible, and the generated SQL follows ANSI standards where it is possible.

The following are considerations for automatic data type conversions:

- `NULL` is specified as `"NULL"`.
- Single quotation marks are converted into double quotation marks for the following data types when they are inline values: `CHAR`, `VARCHAR2`, `NVARCHAR2`, `NCHAR`, `CLOB`, and `NCLOB`.
- `LONG` data is converted into `CLOB` data.
- `LONG RAW` data is converted into `BLOB` data.

SQL Generation and LOB, LONG, LONG RAW, and XMLType Data Types

For `INSERT` and `UPDATE` operations on LOB columns, an outbound server automatically assembles the LOB chunks using LOB assembly.

For these operations, the generated SQL includes a non-`NULL` empty value. The actual values of the chunked columns arrive in subsequent LCRs. For each chunk, you must perform the correct SQL operation on the correct column.

Similarly, for `LONG`, `LONG RAW`, and `XMLType` data types, an outbound server generates a non-`NULL` empty value, and the actual values of the column arrive in chunks in subsequent LCRs. For each chunk, you must perform the correct SQL operation on the correct column.

In the inline version of the generated SQL, for LOB, `LONG`, `LONG RAW`, and `XMLType` data type columns, the following SQL is generated for inserts and updates:

- For `CLOB`, `NCLOB`, and `LONG` data type columns:

  ```
  EMPTY_CLOB()
  ```

- For `BLOB` and `LONG RAW` data type columns:

  ```
  EMPTY_BLOB()
  ```

- For `XMLType` columns:

  ```
  XMLTYPE.CREATEXML('xml /')
  ```

  where *`xml /`* is the XML chunk.

After the LCR that contains the DML statement arrives, the data for these changes arrive in separate chunks. You can generate the `WHERE` clause for such a change and use the generated `WHERE` clause to identify the row for the modifications contained in the chunks. For example, in PL/SQL you can use the `GET_WHERE_CLAUSE` row LCR member procedure to generate the `WHERE` clause for a row change.

For `INSERT` and `UPDATE` operations, the generated `WHERE` clause identifies the row after the insert or update. For example, consider the following update to the `hr.departments` table:

```
UPDATE hr.departments SET department_name='Management'
  WHERE department_name='Administration';
```

The generated `WHERE` clause for this change is the following:

```
WHERE "DEPARTMENT_NAME"='Management'
```

For piecewise LOB operation performed by subprograms in the `DBMS_LOB` package (including the `WRITE`, `TRIM`, and `ERASE` procedures), the generated SQL includes a `SELECT FOR UPDATE` statement.

For example, a `LOB_WRITE` operation on a `clob_col` results in generated SQL similar to the following:

```
SELECT "CLOB_COL" FROM "HR"."LOB_TAB" WHERE "N1"=2 FOR UPDATE
```

The selected `clob_col` must be defined. You can use the LOB locator to perform piecewise LOB operations with the LOB chunks that follow the row LCR.

See Also:

"Sample Generated SQL Statements"

SQL Generation and Character Sets

When you use the LCR methods, the generated SQL is in the database character set.

SQL keywords, such as `INSERT`, `UPDATE`, and `INTO`, do not change with the character set.

See Also:

- *Oracle Database Globalization Support Guide* for information about data conversion in JDBC
- *Oracle Database SQL Language Reference* for information about SQL keywords

Sample Generated SQL Statements

Examples illustrate generated SQL statements.

- Sample Generated SQL Statements for the hr.employees Table
  Examples illustrate SQL statements generated by an outbound server for changes made to the `hr.employees` table.
- Sample Generated SQL Statements for a Table With LOB Columns
  Examples illustrate SQL statements generated by an outbound server for changes made ti a table with LOB columns.

Sample Generated SQL Statements for the hr.employees Table

Examples illustrate SQL statements generated by an outbound server for changes made to the `hr.employees` table.

> **Note:**
>
> Generated SQL is in a single line and is not formatted.

Example 2-1   Generated Insert

Assume the following insert is executed:

```
INSERT INTO hr.employees (employee_id,
                           last_name,
                           email,
                           hire_date,
                           job_id,
                           salary,
                           commission_pct)
                   VALUES (207,
                           'Gregory',
                           'pgregory@example.com',
                           SYSDATE,
                           'PU_CLERK',
                           9000,
                           NULL);
```

The following is the generated SQL with inline values:

```
INSERT INTO "HR"."EMPLOYEES"("EMPLOYEE_ID","FIRST_NAME","LAST_NAME",
"EMAIL","PHONE_NUMBER","HIRE_DATE","JOB_ID","SALARY","COMMISSION_PCT",
"MANAGER_ID","DEPARTMENT_ID" ) VALUES ( 207, NULL,'Gregory',
'pgregory@example.com', NULL , TO_DATE(' 2009-04-15','syyyy-mm-dd'),
'PU_CLERK',9000, NULL , NULL , NULL )
```

The following is the generated SQL with bind variables:

```
INSERT INTO "HR"."EMPLOYEES"("EMPLOYEE_ID","FIRST_NAME","LAST_NAME",
"EMAIL","PHONE_NUMBER","HIRE_DATE","JOB_ID","SALARY",
"COMMISSION_PCT","MANAGER_ID","DEPARTMENT_ID" ) VALUES ( :1   ,:2   ,:3
,:4   ,:5   ,:6   ,:7   ,:8   ,:9   ,:10  ,:11  )
```

Example 2-2   Generated Update

Assume the following update is executed:

```
UPDATE hr.employees SET salary=10000 WHERE employee_id=207;
```

The following is the generated SQL with inline values:

```
UPDATE "HR"."EMPLOYEES" SET "SALARY"=10000 WHERE "EMPLOYEE_ID"=207
AND "SALARY"=9000
```

The following is the generated SQL with bind variables:

```
UPDATE "HR"."EMPLOYEES" SET "SALARY"=:1    WHERE "EMPLOYEE_ID"=:2
AND "SALARY"=:3
```

Example 2-3   Generated Delete

Assume the following delete is executed:

```
DELETE FROM hr.employees WHERE employee_id=207;
```

The following is the generated SQL with inline values:

```
DELETE  FROM "HR"."EMPLOYEES" WHERE "EMPLOYEE_ID"=207 AND "FIRST_NAME" IS NULL
AND "LAST_NAME"='Gregory' AND "EMAIL"='pgregory@example.com' AND
"PHONE_NUMBER" IS NULL  AND "HIRE_DATE"= TO_DATE(' 2009-04-15','syyyy-mm-dd')
AND "JOB_ID"='PU_CLERK' AND "SALARY"=10000 AND "COMMISSION_PCT" IS NULL
AND "MANAGER_ID" IS NULL  AND "DEPARTMENT_ID" IS NULL
```

The following is the generated SQL with bind variables:

```
DELETE  FROM "HR"."EMPLOYEES" WHERE "EMPLOYEE_ID"=:1    AND "FIRST_NAME"=:2
AND "LAST_NAME"=:3    AND "EMAIL"=:4    AND "PHONE_NUMBER"=:5    AND
"HIRE_DATE"=:6    AND "JOB_ID"=:7    AND "SALARY"=:8    AND
"COMMISSION_PCT"=:9   AND "MANAGER_ID"=:10   AND "DEPARTMENT_ID"=:11
```

Sample Generated SQL Statements for a Table With LOB Columns

Examples illustrate SQL statements generated by an outbound server for changes made ti a table with LOB columns.

Examples illustrate SQL statements generated by an outbound server for changes made to the following table:

```
CREATE TABLE hr.lob_tab(
   n1         number primary key,
   clob_col   CLOB,
   nclob_col  NCLOB,
   blob_col   BLOB);
```

Note:

Generated SQL is in a single line and is not formatted.

The `GET_WHERE_CLAUSE` member procedure generates the following `WHERE` clause for this insert:

- Inline:

  ```
  WHERE "N1"=2
  ```

- Bind variables:

  ```
  WHERE "N1"=:1
  ```

You can use the `WHERE` clause to identify the row that was inserted when the subsequent chunks arrive for the LOB column change.

Example 2-4   Generated Insert for a Table with LOB Columns

Assume the following insert is executed:

```
INSERT INTO hr.lob_tab VALUES (2, 'test insert', NULL, NULL);
```

The following is the generated SQL with inline values:

```
INSERT INTO "HR"."LOB_TAB"("N1","BLOB_COL","CLOB_COL","NCLOB_COL" )
VALUES ( 2,, EMPTY_CLOB() ,)
```

The following is the generated SQL with bind variables:

```
INSERT INTO "HR"."LOB_TAB"("N1","BLOB_COL","CLOB_COL","NCLOB_COL" )
VALUES ( :1   ,:2   ,:3   ,:4   )
```

Example 2-5 Generated Update for a Table with LOB Columns

Assume the following update is executed:

```
UPDATE hr.lob_tab SET clob_col='test update' WHERE n1=2;
```

The following is the generated SQL with inline values:

```
UPDATE "HR"."LOB_TAB" SET "CLOB_COL"= EMPTY_CLOB()  WHERE "N1"=2
```

The following is the generated SQL with bind variables:

```
UPDATE "HR"."LOB_TAB" SET "CLOB_COL"=:1    WHERE "N1"=:2
```

Example 2-6 Generated Delete for a Table with LOB Columns

Assume the following delete is executed:

```
DELETE FROM hr.lob_tab WHERE n1=2;
```

The following is the generated SQL with inline values:

```
DELETE  FROM "HR"."LOB_TAB" WHERE "N1"=2
```

The following is the generated SQL with bind variables:

```
DELETE  FROM "HR"."LOB_TAB" WHERE "N1"=:1
```

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 Replication 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.

Part II
XStream Out

You can configure and manage an XStream Out environment.

- XStream Out Concepts
  Become familiar with concepts related to XStream Out.
- Configuring XStream Out
  You can configure the Oracle Database components that are used by XStream Out.
- Managing XStream Out
  You can manage XStream Out components and their rules.
- Monitoring XStream Out
  You can monitor an XStream Out configuration.
- Troubleshooting XStream Out
  You can diagnose and correct problems with an XStream Out configuration.

3

XStream Out Concepts

Become familiar with concepts related to XStream Out.

- Introduction to XStream Out
  XStream Out can capture transactions from the redo log of an Oracle database and send them efficiently to a client application.
- Capture Processes
  Become familiar with concepts related to capture processes.
- Outbound Servers
  With XStream Out, an outbound server sends database changes to a client application.
- Position of LCRs and XStream Out
  An XStream Out outbound server streams LCRs that were captured by a capture process to a client application. The position of an LCR identifies its placement in the stream of LCRs in a transaction.
- XStream Out and Distributed Transactions
  There are considerations for XStream Out and distributed transactions.
- XStream Out and Security
  Understand security related to the client application and XStream components, as well as the privileges required by the capture user and the connect user.
- XStream Out and Other Oracle Database Components
  XStream Out can work with other Oracle Database components.

See Also:

Configuring XStream Out

Introduction to 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.

In an XStream Out configuration, a capture process captures database changes and sends these changes to an outbound server. This section describes capture processes and outbound servers in detail.

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

See Also:

- *Oracle Call Interface Developer's Guide*
- *Oracle Database XStream Java API Reference*

Capture Processes

Become familiar with concepts related to capture processes.

- Capture Process Overview
  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.
- Data Types Captured by a Capture Process
  A capture process can capture changes made to columns of most data types.
- Types of DML Changes Captured by Capture Processes
  A capture process can capture different types of DML changes.
- Tables, Views, and Materialized Views
- Local Capture and Downstream Capture
  You can configure a capture process to run locally on a source database or remotely on a downstream database.
- Capture Processes and RESTRICTED SESSION
  Enabling and disabling restricted session affects capture processes.
- XStream Out Process Subcomponents
  The XStream Out process subcomponents are a reader server, one or more preparer servers, and a builder server.
- Capture Process States
  The state of a capture process describes what the capture process is doing currently.
- Capture Process Parameters
  Capture process parameters control the way a capture process operates.
- Capture Process Checkpoints and XStream Out
  A checkpoint is information about the current state of a capture process that is stored persistently in the data dictionary of the database running the capture process.
- SCN Values Related to a Capture Process
  Specific system change number (SCN) values are important for a capture process.

Capture Process Overview

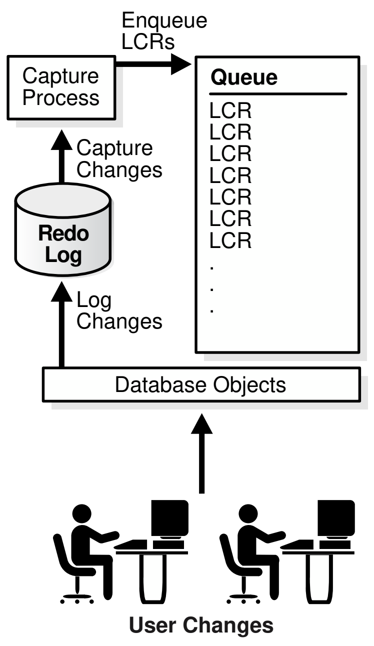
A **capture process** is an optional Oracle background process that scans the database redo log to capture DML and DDL changes made to database objects.

The primary function of the redo log is to record all of the changes made to the database. A capture process captures database changes from the redo log, and the database where the changes were generated is called the **source database** for the capture process.

When a capture process captures a database change, it converts it into a specific message format called a logical change record (LCR). In an XStream Out configuration, the capture process sends these LCRs to an outbound server.

Figure 3-1 shows a capture process.

Figure 3-1 Capture Process



A capture process can run on its source database or on a remote database. When a capture process runs on its source database, the capture process is a **local capture process**.

You can also capture changes for the source database by running the capture process on different server. When a capture process runs on a remote database, the capture process is called a **downstream capture process**, and the remote database is called the **downstream database**. The log files are written to the remote database and to the source database. In this configuration, the source logfiles must be available at the downstream capture database. The capture process on the remote database mines the logs from the source database and stages them locally. This configuration can be helpful when you want to offload the processing of capture changes from a production database to different, remote database.

Data Types Captured by a Capture Process

A capture process can capture changes made to columns of most data types.

When capturing the row changes resulting from DML changes made to tables, a capture process can capture changes made to columns of the following data types:

- `NUMBER`
- `FLOAT`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `VARCHAR2`
- `NVARCHAR2`
- `CHAR`

- `NCHAR`
- `BOOLEAN`
- `DATE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `LONG`
- `LONG RAW`
- `UROWID`
- `CLOB` with `BASICFILE` or `SECUREFILE` storage
- `NCLOB` with `BASICFILE` or `SECUREFILE` storage
- `BLOB` with `BASICFILE` or `SECUREFILE` storage
- `BFILE`
- `XMLType` stored as `CLOB`, object relational, or as binary XML
- `JSON`
- Varrays
- `REF` data types

Types of DML Changes Captured by Capture Processes

A capture process can capture different types of DML changes.

When you specify that DML changes made to certain tables should be captured, a capture process captures the following types of DML changes made to these tables:

- `INSERT`
- `UPDATE`
- `DELETE`
- `MERGE`
- Piecewise operations

A capture process converts each `MERGE` change into an `INSERT` or `UPDATE` change. `MERGE` is not a valid command type in a row LCR.

If XStream is replicating data for an object type, then the replication must be unidirectional, and all replication sites must agree on the names and data types of the attributes in the object type. You establish the names and data types of the attributes when you create the object type. For example, consider the following object type:

```
CREATE TYPE cust_address_typ AS OBJECT
       (street_address     VARCHAR2(40),
        postal_code        VARCHAR2(10),
        city              VARCHAR2(30),
        state_province    VARCHAR2(10),
        country_id        CHAR(2));
/
```

At all replication sites, `street_address` must be `VARCHAR2(40)`, `postal_code` must be `VARCHAR2(10)`, `city` must be `VARCHAR2(30)`, and so on.

Note:

- The maximum size of the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types has been increased in Oracle Database 12*c* when the `COMPATIBLE` initialization parameter is set to `12.0.0` and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`.
- `XMLType` stored as a `CLOB` is deprecated in Oracle Database 12*c* Release 1 (12.1).
- For `BFILE`, only the data type structure is replicated and not the content of the `BFILE` that exists on the file system.

- ID Key LCRs
  An **ID key LCR** is a special type of row LCR. ID key LCRs enable an XStream client application to process changes to rows that include unsupported data types.
- ID Key LCRs Demo
  A demo is available that creates a sample client application that processes ID key LCRs.

ID Key LCRs

An **ID key LCR** is a special type of row LCR. ID key LCRs enable an XStream client application to process changes to rows that include unsupported data types.

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

- `ROWID`
- Nested tables
- The following Oracle-supplied types: `ANYTYPE`, `ANYDATASET`, URI types, `SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`, and `Expression`.

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

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 data for columns of unsupported data types.

XStream Out can capture changes for tables that are not fully supported by setting the `CAPTURE_IDKEY_OBJECTS` capture process parameter to `Y`. 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.

You can use the `DBA_XSTREAM_OUT_SUPPORT_MODE` view to display a list of local tables supported by ID key LCRs. This view shows the following types of XStream Out support for tables in the `SUPPORT_MODE` column:

- `FULL` for tables that are fully supported by XStream Out (without using ID key LCRs)
- `ID KEY` for tables that are supported only by using ID key LCRs
- `NONE` for tables that are not supported by XStream Out.

  Even ID key LCRs cannot be used to process changes to rows in tables that show `NONE` in the `DBA_XSTREAM_OUT_SUPPORT_MODE` view.

For example, run the following query to show XStream Out support for all of the tables in the database:

```
COLUMN OWNER FORMAT A30
COLUMN OBJECT_NAME FORMAT A30
COLUMN SUPPORT_MODE FORMAT A12

SELECT OWNER, OBJECT_NAME, SUPPORT_MODE
  FROM DBA_XSTREAM_OUT_SUPPORT_MODE
  ORDER BY OBJECT_NAME;
```

Your output is similar to the following:

```
OWNER                          OBJECT_NAME                    SUPPORT_MODE
------------------------------ ------------------------------ ------------
.
.
.

IX                             ORDERS_QUEUETABLE              NONE
OE                             ORDER_ITEMS                    FULL
SH                             PLAN_TABLE                     FULL
PM                             PRINT_MEDIA                    ID KEY
.
.
.
```

See Also:

- *Oracle Database Reference*
- "Row LCRs"
- "Setting a Capture Process Parameter"

ID Key LCRs Demo

A demo is available that creates a sample client application that processes 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
```

Tables, Views, and Materialized Views

The following DML operations are supported for regular tables, index-organized tables, clustered tables, and materialized views:

- `INSERT`
- `UPDATE`
- `DELETE`
- Associated transaction control operations

Starting with Oracle Database 23ai, the following features are available for tables:

- 4K column in tables with row size less than 4 MB
- Blockchain and Immutable Tables

  See Scope of Support for Blockchain and Immutable Tables for details.

Tip:

You can use the `DBA_XSTREAM_SUPPORT_MODE` data dictionary view to display information about the level of Oracle GoldenGate capture process support for the tables in your database. The `PLSQL` value of `DBA_XSTREAM_SUPPORT_MODE` indicates that the table is supported natively, but requires procedural supplemental logging. For more information, see the `DBA_XSTREAM_SUPPORT_MODE`.

- Scope of Support for Lock-free Reservation
- Scope of Support for Blockchain and Immutable Tables

Scope of Support for Lock-free Reservation

Starting with Oracle Database 23ai, Oracle XStream-Out supports lock-free reservable columns.

Source tables using lock-free columns to the target tables *with* implicit delta conflict resolution for replication from reservable columns on the source to the reservable columns on the target.

Scope of Support for Blockchain and Immutable Tables

Starting with Oracle Database 23ai, XStream-Out supports capturing changes from blockchain and immutable tables.

Local Capture and Downstream Capture

You can configure a capture process to run locally on a source database or remotely on a downstream database.

A single database can have one or more capture processes that capture local changes and other capture processes that capture changes from a remote source database. That is, you can configure a single database to perform both local capture and downstream capture.

- Local Capture
  Local capture means that a capture process runs on the source database.
- Downstream Capture

Local Capture

Local capture means that a capture process runs on the source database.

Figure 3-1 shows a database using local capture.

- The Source Database Performs All Change Capture Actions
  With local capture, the capture actions are performed at the source database.
- Advantages of Local Capture
  Local capture has several advantages.

The Source Database Performs All Change Capture Actions

With local capture, the capture actions are performed at the source database.

If you configure local capture, then the following actions are performed at the source database:

- The `DBMS_CAPTURE_ADM.BUILD` procedure is run to extract (or build) the data dictionary to the redo log.
- Supplemental logging at the source database places additional information in the redo log. This information might be needed when captured changes are processed by an XStream client application. See "If Required, Configure Supplemental Logging".
- The first time a capture process is started at the database, Oracle Database uses the extracted data dictionary information in the redo log to create a LogMiner data dictionary, which is separate from the primary data dictionary for the source database. Additional capture processes can use this existing LogMiner data dictionary, or they can create new LogMiner data dictionaries.
- A capture process scans the redo log for changes using LogMiner.
- The rules engine evaluates changes based on the rules in one or more of the capture process rule sets.
- The capture process enqueues changes that satisfy the rules in its rule sets into a local `ANYDATA` queue.
- If the captured changes are shared with one or more outbound servers on other databases, then one or more propagations propagate these changes from the source database to the other databases.

Advantages of Local Capture

Local capture has several advantages.

The following are the advantages of using local capture:

- Configuration and administration of the capture process is simpler than when downstream capture is used. When you use local capture, you do not need to configure redo data copying to a downstream database, and you administer the capture process locally at the database where the captured changes originated.
- A local capture process can scan changes in the online redo log before the database writes these changes to an archived redo log file. When you use an archived-log downstream capture process, archived redo log files are copied to the downstream database after the source database has finished writing changes to them, and some time is required to copy the redo log files to the downstream database. However, a real-time downstream capture process can capture changes in the online redo log sent from the source database.
- The amount of data being sent over the network is reduced, because the redo data is not copied to the downstream database. Even if captured LCRs are propagated to other databases, the captured LCRs can be a subset of the total changes made to the database, and only the LCRs that satisfy the rules in the rule sets for a propagation are propagated.
- Security might be improved because only the source (local) database can access the redo data. For example, if the capture process captures changes in the hr schema only, then, when you use local capture, only the source database can access the redo data to enqueue changes to the hr schema into the capture process queue. However, when you use downstream capture, the redo data is copied to the downstream database, and the redo data contains all of the changes made to the database, not just the changes made to a specific object or schema.

Downstream Capture

The following types of downstream capture configurations are possible: real-time downstream capture and archived-log downstream capture. The `downstream_real_time_mine` capture process parameter controls whether a downstream capture process performs real-time downstream capture or archived-log downstream capture. A real-time downstream capture process and one or more archived-log downstream capture processes can coexist at a downstream database. With downstream capture, the redo log files of the source database must be available at the downstream database.

Note:

- References to "downstream capture processes" in this document apply to both real-time downstream capture processes and archived-log downstream capture processes. This document distinguishes between the two types of downstream capture processes when necessary.
- Per PDB XStream Out does not exist in Downstream Capture.
- A downstream capture process only can capture changes from a single source database. However, multiple downstream capture processes at a single downstream database can capture changes from a single source database or multiple source databases.

- Real-Time Downstream Capture
  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 at the source database.

- Archived-Log Downstream Capture
  The advantage of archived-log downstream capture over real-time downstream capture is that archived-log downstream capture allows downstream capture processes from multiple source databases at a downstream database.
- The Downstream Database Performs Most Change Capture Actions
  With downstream capture, most capture actions are performed at the downstream database.
- Advantages of Downstream Capture
  Downstream capture provides several advantages.
- Optional Database Link From the Downstream Database to the Source Database
  When you create or alter a downstream capture process, you optionally can specify the use of a database link from the downstream database to the source database.
- Operational Requirements for Downstream XStream Out with XStream Out
  Some operational requirements apply to downstream XStream Out.

Real-Time Downstream Capture

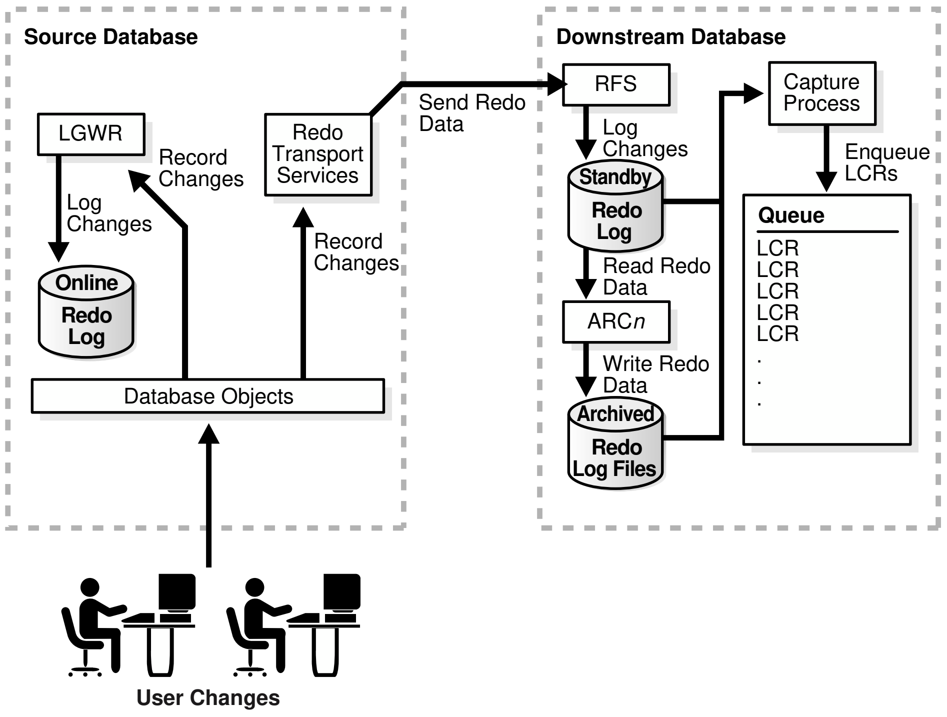
The advantage of real-time downstream capture over archived-log downstream capture is that real-time downstream capture reduces the amount of time required to capture changes made at the source database.

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 data from it.

A real-time downstream capture configuration works in the following way:

- Redo transport services sends redo data to the downstream database either synchronously or asynchronously. At the same time, the log writer process (LGWR) records redo data in the online redo log at the source database.
- 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.
- A log switch at the source database causes a log switch at the downstream database, and the `ARCH`*n* process at the downstream database archives the current standby redo log file.
- The real-time downstream capture process captures changes from the standby redo log whenever possible and from the archived standby redo log files whenever necessary. A capture process can capture changes in the archived standby redo log files if it falls behind. When it catches up, it resumes capturing changes from the standby redo log.

Figure 3-2   Real-Time Downstream Capture



Note:

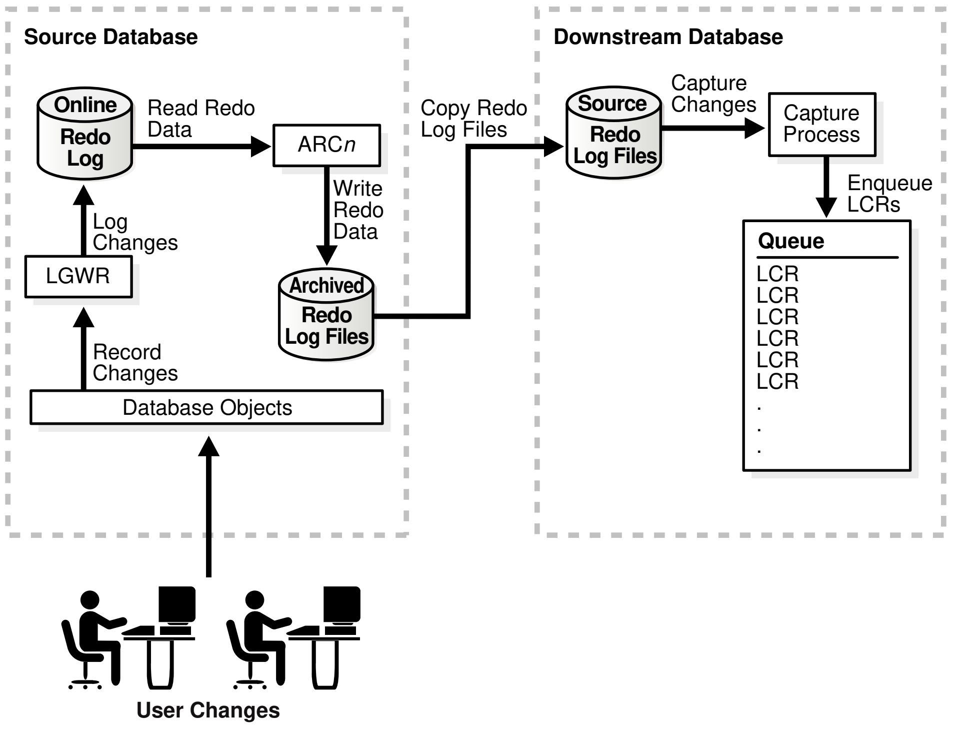
You can configure more than one real-time downstream capture process that captures changes from the same source database, but you cannot configure real-time downstream capture for multiple source databases at one downstream database.

Archived-Log Downstream Capture

The advantage of archived-log downstream capture over real-time downstream capture is that archived-log downstream capture allows downstream capture processes from multiple source databases at a downstream database.

An archived-log downstream capture 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. You can copy the archived redo log files to the downstream database using redo transport services, the `DBMS_FILE_TRANSFER` package, file transfer protocol (FTP), or some other mechanism.

Figure 3-3 Archived-Log Downstream Capture



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 Data Guard Concepts and Administration for more information about redo transport services

The Downstream Database Performs Most Change Capture Actions

With downstream capture, most capture actions are performed at the downstream database.

If you configure either real-time or archived-log downstream capture, then the following actions are performed at the downstream database:

- The first time a downstream capture process is started at the downstream database, Oracle Database uses data dictionary information in the redo data from the source database to create a LogMiner data dictionary at the downstream database. The `DBMS_CAPTURE_ADM.BUILD` procedure is run at the source database to extract the source data dictionary information to the redo log at the source database. Next, the redo data is copied to the downstream database from the source database. Additional downstream capture processes for the same source database can use this existing LogMiner data dictionary, or they can create new LogMiner data dictionaries. Also, a real-time downstream capture process can share a LogMiner data dictionary with one or more archived-log downstream capture processes.

- A capture process scans the redo data from the source database for changes using LogMiner.
- The rules engine evaluates changes based on the rules in one or more of the capture process rule sets.
- The capture process enqueues changes that satisfy the rules in its rule sets into a local `ANYDATA` queue. The capture process formats the changes as LCRs.

In a downstream capture configuration, the following actions are performed at the source database:

- The `DBMS_CAPTURE_ADM.BUILD` procedure is run at the source database to extract the data dictionary to the redo log.
- Supplemental logging at the source database places additional information that might be needed for apply in the redo log. See "If Required, Configure Supplemental Logging".

In addition, the redo data must be copied from the computer system running the source database to the computer system running the downstream database. In a real-time downstream capture configuration, redo transport services sends redo data to the downstream database. Typically, in an archived-log downstream capture configuration, redo transport services copies the archived redo log files to the downstream database.

Advantages of Downstream Capture

Downstream capture provides several advantages.

The following are the advantages of using downstream capture:

- Capturing changes uses fewer resources at the source database because the downstream database performs most of the required work.
- If you plan to capture changes originating at multiple source databases, then capture process administration can be simplified by running multiple archived-log downstream capture processes with different source databases at one downstream database. That is, one downstream database can act as the central location for change capture from multiple sources. In such a configuration, one real-time downstream capture process can run at the downstream database in addition to the archived-log downstream capture processes.
- Copying redo data to one or more downstream databases provides improved protection against data loss. For example, redo log files at the downstream database can be used for recovery of the source database in some situations.
- The ability to configure at one or more downstream databases multiple capture processes that capture changes from a single source database provides more flexibility and can improve scalability.

Optional Database Link From the Downstream Database to the Source Database

When you create or alter a downstream capture process, you optionally can specify the use of a database link from the downstream database to the source database.

This database link must have the same name as the global name of the source database. Such a database link simplifies the creation and administration of a downstream capture process. You specify that a downstream capture process uses a database link by setting the `use_database_link` parameter to `TRUE` when you run the `CREATE_CAPTURE` or `ALTER_CAPTURE` procedure on the downstream capture process. The name of the database link must match the global name of the source database.

When a downstream capture process uses a database link to the source database, the capture process connects to the source database to perform the following administrative actions automatically:

- In certain situations, runs the `DBMS_CAPTURE_ADM.BUILD` procedure at the source database to extract the data dictionary at the source database to the redo log when a capture process is created.
- Obtains the first SCN for the downstream capture process if the first system change number (SCN) is not specified during capture process creation. The first SCN is needed to create a capture process.

If a downstream capture process does not use a database link, then you must perform these actions manually.

Note:

During the creation of a downstream capture process, if the `first_scn` parameter is set to `NULL` in the `CREATE_CAPTURE` procedure, then the use_database_link parameter must be set to `TRUE`. Otherwise, an error is raised.

Operational Requirements for Downstream XStream Out with XStream Out

Some operational requirements apply to downstream XStream Out.

The following are operational requirements for using downstream XStream Out:

- The source database must be running at least Oracle Database 12*c* Release 2 (12.2).
- The operating system on the source and downstream XStream Out sites must be the same, but the operating system release does not need to be the same. In addition, the downstream sites can use a directory structure that is different from the source site.
- The hardware architecture on the source and downstream XStream Out sites must be the same. For example, a downstream XStream Out configuration with a source database on a 64-bit Sun system must have a downstream database that is configured on a 64-bit Sun system. Other hardware elements, such as the number of CPUs, memory size, and storage configuration, can differ in the source and downstream sites.

Note:

- A sourceless cascaded XStream setup is not supported.
- A DownStream XStream is not supported in a cascaded environment.

Capture Processes and RESTRICTED SESSION

Enabling and disabling restricted session affects capture processes.

When you enable restricted session during system startup by issuing a `STARTUP RESTRICT` statement, capture processes do not start, even if they were running when the database shut down. When restricted session is disabled with an `ALTER SYSTEM` statement, each capture process that was running when the database shut down is started.

When restricted session is enabled in a running database by the SQL statement `ALTER SYSTEM ENABLE RESTRICTED SESSION` clause, it does not affect any running capture processes. These capture processes continue to run and capture changes. If a stopped capture process is started in a restricted session, then the capture process does not actually start until the restricted session is disabled.

XStream Out Process Subcomponents

The XStream Out process subcomponents are a reader server, one or more preparer servers, and a builder server.

A XStream Out process is an optional Oracle background process whose process name is `CP`*`nn`*, where *`nn`* can include letters and numbers. A XStream Out process XStream Outs changes from the redo log by using the infrastructure of LogMiner. XStream configures LogMiner automatically. You can create, alter, start, stop, and drop a XStream Out process, and you can define XStream Out process rules that control which changes a XStream Out process XStream Outs.

The `parallelism` XStream Out process parameter controls XStream Out process parallelism. When XStream Out process parallelism is 0 (zero), the default for XStream Out, the XStream Out process does not use subcomponents to perform its work. Instead, the `CP`*`nn`* process completes all of the tasks required to XStream Out database changes.

When XStream Out process parallelism is greater than 0, the XStream Out process uses the underlying LogMiner process name is `MS`*`nn`*, where *`nn`* can include letters and numbers. When XStream Out process parallelism is 0 (zero), the XStream Out process does not use this process.

When XStream Out process parallelism is greater than 0, the XStream Out process consists of the following subcomponents:

- One **reader server** that reads the redo log and divides the redo log into regions.
- One or more **preparer servers** that scan the regions defined by the reader server in parallel and perform prefiltering of changes found in the redo log. Prefiltering involves sending partial information about changes, such as schema and object name for a change, to the rules engine for evaluation, and receiving the results of the evaluation. You can control the number of preparer servers using the parallelism XStream Out process parameter.
- One **builder server** that merges redo records from the preparer servers. These redo records either evaluated to `TRUE` during partial evaluation or partial evaluation was inconclusive for them. The builder server preserves the system change number (SCN) order of these redo records and passes the merged redo records to the XStream Out process.
- The XStream Out process (`CP`*`nn`*) performs the following actions for each change when it receives merged redo records from the builder server:
  - Formats the change into an LCR
  - If the partial evaluation performed by a preparer server was inconclusive for the change in the LCR, then sends the LCR to the rules engine for full evaluation
  - Receives the results of the full evaluation of the LCR if it was performed
  - Discards the LCR if it satisfies the rules in the negative rule set for the XStream Out process or if it does not satisfy the rules in the positive rule set
  - Enqueues the LCR into the queue associated with the XStream Out process if the LCR satisfies the rules in the positive rule set for the XStream Out process

Each reader server, preparer server, and builder server is a process.

Capture Process States

The state of a capture process describes what the capture process is doing currently.

You can view the state of a capture process by querying the `STATE` column in the `V$XSTREAM_CAPTURE` dynamic performance view.

See Also:

Oracle Database Reference

Capture Process Parameters

Capture process parameters control the way a capture process operates.

For example, the parallelism capture process parameter controls the number of preparer servers used by a capture process, and the `time_limit` capture process parameter specifies the amount of time a capture process runs before it is shut down automatically. You set capture process parameters using the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure. After creation, a capture process is disabled so that you can set the capture process parameters for your environment before starting it for the first time.

See Also:

Oracle Database PL/SQL Packages and Types Reference

Capture Process Checkpoints and XStream Out

A checkpoint is information about the current state of a capture process that is stored persistently in the data dictionary of the database running the capture process.

A capture process tries to record a checkpoint at regular intervals called checkpoint intervals.

- Required Checkpoint SCN
  The system change number (SCN) that corresponds to the lowest checkpoint for which a capture process requires redo data is the required checkpoint SCN.
- Maximum Checkpoint SCN
  The SCN that corresponds to the last physical checkpoint recorded by a capture process is the maximum checkpoint SCN.
- Checkpoint Retention Time
  The checkpoint retention time is the amount of time, in number of days, that a capture process retains checkpoints before purging them automatically.

Required Checkpoint SCN

The system change number (SCN) that corresponds to the lowest checkpoint for which a capture process requires redo data is the required checkpoint SCN.

The redo log file that contains the required checkpoint SCN, and all subsequent redo log files, must be available to the capture process. If a capture process is stopped and restarted, then it starts scanning the redo log from the SCN that corresponds to its required checkpoint SCN. The required checkpoint SCN is important for recovery if a database stops unexpectedly. Also, if the first SCN is reset for a capture process, then it must be set to a value that is less than or equal to the required checkpoint SCN for the captured process. You can determine the required checkpoint SCN for a capture process by querying the `REQUIRED_CHECKPOINT_SCN` column in the `ALL_CAPTURE` data dictionary view.

Maximum Checkpoint SCN

The SCN that corresponds to the last physical checkpoint recorded by a capture process is the maximum checkpoint SCN.

The maximum checkpoint SCN can be lower than or higher than the required checkpoint SCN for a capture process. The maximum checkpoint SCN can also be 0 (zero) if the capture process is new and has not yet recorded a physical checkpoint.

Checkpoint Retention Time

The checkpoint retention time is the amount of time, in number of days, that a capture process retains checkpoints before purging them automatically.

A capture process periodically computes the age of a checkpoint by subtracting the `NEXT_TIME` of the archived redo log file that corresponds to the checkpoint from `FIRST_TIME` of the archived redo log file containing the required checkpoint SCN for the capture process. If the resulting value is greater than the checkpoint retention time, then the capture process automatically purges the checkpoint by advancing its first SCN value. Otherwise, the checkpoint is retained.

You can use the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to set the checkpoint retention time for a capture process. The `DBA_REGISTERED_ARCHIVED_LOG` view displays the `FIRST_TIME` and `NEXT_TIME` for archived redo log files, and the `REQUIRED_CHECKPOINT_SCN` column in the `ALL_CAPTURE` view displays the required checkpoint SCN for a capture process.

See Also:

Oracle Database PL/SQL Packages and Types Reference

SCN Values Related to a Capture Process

Specific system change number (SCN) values are important for a capture process.

You can query the `ALL_CAPTURE` data dictionary view to display these values for one or more capture processes.

- Captured SCN and Applied SCN
  The captured SCN is the SCN that corresponds to the most recent change scanned in the redo log by a capture process. The applied SCN for a capture process is the SCN of the most recent LCR processed by the relevant outbound server.
- First SCN and Start SCN
  The first SCN and start SCN are important for a capture process.

Captured SCN and Applied SCN

The captured SCN is the SCN that corresponds to the most recent change scanned in the redo log by a capture process. The applied SCN for a capture process is the SCN of the most recent LCR processed by the relevant outbound server.

All LCRs lower than the applied SCN have been processed by all outbound servers that process changes captured by the capture process. The applied SCN for a capture process is equivalent to the low-watermark SCN for an outbound server that processes changes captured by the capture process.

First SCN and Start SCN

The first SCN and start SCN are important for a capture process.

- First SCN
  The first SCN is the lowest SCN in the redo log from which a capture process can capture changes.
- Start SCN
  The start SCN is the SCN from which a capture process begins to capture changes.
- Start SCN Must Be Greater Than or Equal to First SCN
  If you specify a start SCN when you create or alter a capture process, then the start SCN specified must be greater than or equal to the first SCN for the capture process.

First SCN

The first SCN is the lowest SCN in the redo log from which a capture process can capture changes.

If you specify a first SCN during capture process creation, then the database must be able to access redo data from the SCN specified and higher.

The DBMS_CAPTURE_ADM.BUILD procedure extracts the source database data dictionary to the redo log. When you create a capture process, you can specify a first SCN that corresponds to this data dictionary build in the redo log. Specifically, the first SCN for the capture process being created can be set to any value returned by the following query:

```
COLUMN FIRST_CHANGE# HEADING 'First SCN' FORMAT 999999999
COLUMN NAME HEADING 'Log File Name' FORMAT A50

SELECT DISTINCT FIRST_CHANGE#, NAME FROM V$ARCHIVED_LOG
  WHERE DICTIONARY_BEGIN = 'YES';
```

The value returned for the NAME column is the name of the redo log file that contains the SCN corresponding to the first SCN. This redo log file, and all subsequent redo log files, must be available to the capture process. If this query returns multiple distinct values for FIRST_CHANGE#, then the DBMS_CAPTURE_ADM.BUILD procedure has been run more than once on the source database. In this case, choose the first SCN value that is most appropriate for the capture process you are creating.

In some cases, the `DBMS_CAPTURE_ADM.BUILD` procedure is run automatically when a capture process is created. When this happens, the first SCN for the capture process corresponds to this data dictionary build.

Start SCN

The start SCN is the SCN from which a capture process begins to capture changes.

The start SCN is the SCN from which a capture process begins to capture changes. You can specify a start SCN that is different than the first SCN during capture process creation, or you can alter a capture process to set its start SCN. The start SCN does not need to be modified for normal operation of a capture process. Typically, you reset the start SCN for a capture process if point-in-time recovery must be performed on one of the destination databases that receive changes from the capture process. In these cases, the capture process can capture the changes made at the source database after the point-in-time of the recovery.

Note:

An existing capture process must be stopped before setting its start SCN.

Start SCN Must Be Greater Than or Equal to First SCN

If you specify a start SCN when you create or alter a capture process, then the start SCN specified must be greater than or equal to the first SCN for the capture process.

A capture process always scans any unscanned redo log records that have higher SCN values than the first SCN, even if the redo log records have lower SCN values than the start SCN. So, if you specify a start SCN that is greater than the first SCN, then the capture process might scan redo log records for which it cannot capture changes, because these redo log records have a lower SCN than the start SCN.

Scanning redo log records before the start SCN should be avoided if possible because it can take some time. Therefore, Oracle recommends that the difference between the first SCN and start SCN be as small as possible during capture process creation to keep the initial capture process startup time to a minimum.

Note:

When a capture process is started or restarted, it might need to scan redo log files with a `FIRST_CHANGE#` value that is lower than start SCN. Removing required redo log files before they are scanned by a capture process causes the capture process to abort. You can query the `ALL_CAPTURE` data dictionary view to determine the first SCN, start SCN, and required checkpoint SCN. A capture process needs the redo log file that includes the required checkpoint SCN, and all subsequent redo log files.

Outbound Servers

With XStream Out, an outbound server sends database changes to a client application.

- Overview of Outbound Servers
  An **outbound server** is an optional Oracle background process that sends database changes to a client application.
- Data Types Supported by Outbound Servers
  Outbound servers support all of the data types that are supported by capture processes.
- Apply User for an Outbound Server
  The apply user for an outbound server is the user who receives LCRs from the outbound server's capture process.
- Outbound Servers and RESTRICTED SESSION
  Enabling and disabling restricted session affects outbound servers.
- Outbound Server Subcomponents
  An outbound server consists of a reader server, a coordinator process, and an apply server.
- Considerations for Outbound Servers
  There are several considerations for XStream outbound servers.
- Outbound Servers and Apply Parameters
  Apply parameters control the behavior of outbound servers.

Overview of Outbound Servers

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 OCI or Java interfaces.

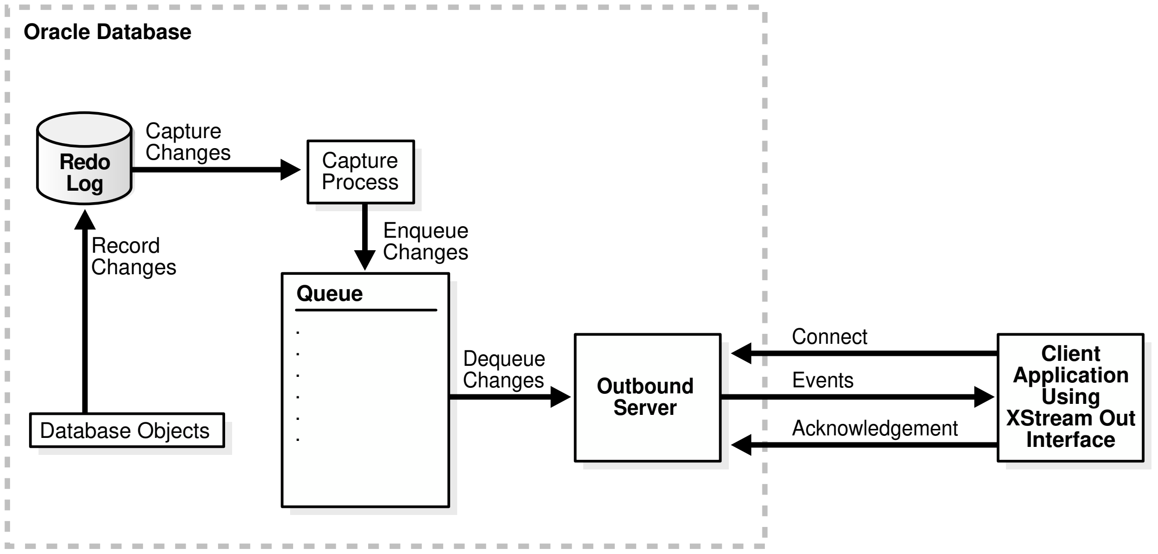
A client application can create multiple sessions. Each session can attach to only one outbound server, and each outbound server can serve only one session at a time. However, different client application sessions can connect to different outbound servers or inbound servers.

When both the outbound server and its capture process 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 3-4 shows an outbound server configuration.

Figure 3-4 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:**
>
> "Capture Processes" for detailed information about capture processes

Data Types Supported by Outbound Servers

Outbound servers support all of the data types that are supported by capture processes.

Outbound servers can send LCRs that include changes to columns of these data types to XStream client applications.

> **See Also:**
>
> "Data Types Captured by a Capture Process"

Apply User for an Outbound Server

The apply user for an outbound server is the user who receives LCRs from the outbound server's capture process.

The apply user for an outbound server must match the capture user for the outbound server's capture process.

See Also:

"Privileges Required by the Capture User for a Capture Process"

Outbound Servers and RESTRICTED SESSION

Enabling and disabling restricted session affects outbound servers.

When restricted session is enabled during system startup by issuing a `STARTUP RESTRICT` statement, outbound servers do not start, even if they were running when the database shut down. When the restricted session is disabled, each outbound server that was not stopped is started.

When restricted session is enabled in a running database by the SQL statement `ALTER SYSTEM ENABLE RESTRICTED SESSION`, it does not affect any running outbound servers. These outbound servers continue to run and send LCRs to an XStream client application. If a stopped outbound server is started in a restricted session, then the outbound server does not actually start until the restricted session is disabled.

Outbound Server Subcomponents

An outbound server consists of a reader server, a coordinator process, and an apply server.

- A **reader server** that receives LCRs from the outbound server's capture process. The reader server is a process that computes dependencies between LCRs and assembles LCRs into transactions. The reader server then returns the assembled transactions to the coordinator process.

  You can view the state of the reader server for an outbound server by querying the `V$XSTREAM_APPLY_READER` dynamic performance view.

- A **coordinator process** that gets transactions from the reader server and passes them to apply servers. The coordinator process name is `AP`*`nn`*, where *`nn`* can include letters and numbers. The coordinator process is an Oracle background process.

  You can view the state of a coordinator process by querying the `V$XSTREAM_APPLY_COORDINATOR` dynamic performance view.

- An **apply server** that sends LCRs to an XStream client application. The apply server is a process. If the apply server encounters an error, then it then it records information about the error in the `ALL_APPLY` view.

  You can view the state of the apply server for an outbound server by querying the `V$XSTREAM_APPLY_SERVER` dynamic performance view.

The reader server and the apply server process names are `AS`*`nn`*, where *`nn`* can include letters and numbers.

See Also:

- *Oracle Database Reference* for more information on `V$XSTREAM_APPLY_READER` dynamic performance view
- *Oracle Database Reference* for more information on `V$XSTREAM_APPLY_COORDINATOR` dynamic performance view
- *Oracle Database Reference* for more information on `V$XSTREAM_APPLY_SERVER` dynamic performance view

Considerations for Outbound Servers

There are several considerations for XStream outbound servers.

The following are considerations for 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 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.
- Automatic split and merge of a stream is possible when the capture process and the outbound server for the stream run on different databases. However, when the capture process and outbound server for a stream run on the same database, automatic split and merge of the stream is not possible.
- An outbound server's LCRs can spill from memory to hard disk if they have been in the buffered queue for a period of time without being processed, if there are a large number of LCRs in a transaction, or if there is not enough space in memory to hold all of the LCRs. An outbound server performs best when a minimum of LCRs spill from memory. You can control an outbound server's behavior regarding spilled LCRs using the `txn_age_spill_threshold` and `txn_lcr_spill_threshold` apply parameters.
- 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 Also:

Oracle Database PL/SQL Packages and Types Reference for information about apply parameters

Outbound Servers and Apply Parameters

Apply parameters control the behavior of outbound servers.

You can use the following apply parameters with outbound servers:

- `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`

See Also:

Oracle Database PL/SQL Packages and Types Reference

Position of LCRs and XStream Out

An XStream Out outbound server streams LCRs that were captured by a capture process to a client application. The position of an LCR identifies its placement in the stream of LCRs in a transaction.

- Additional LCR Attributes Related to Position in XStream Out
  In LCRs that were captured by a capture process, there is additional information related to LCR position.
- The Processed Low Position and Restartability for XStream Out
  The **processed low position** is a position below which all transactions have been processed by the client application.

- 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.

> **See Also:**
>
> "Position Order in an LCR Stream"

Additional LCR Attributes Related to Position in XStream Out

In LCRs that were captured by a capture process, there is additional information related to LCR 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 explicitly captured row LCRs.

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

The Processed Low Position and Restartability for XStream Out

The **processed low position** is a position below which all transactions have been processed by the client application.

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 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"

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 Replication 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.

XStream Out and Distributed Transactions

There are considerations for 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 `11.2.0.0` or higher.

See Also:

- *Oracle Database Administrator's Guide* for more information about distributed transactions
- *Oracle Database Development Guide* for more information about Oracle XA

XStream Out and Security

Understand security related to the client application and XStream components, as well as the privileges required by the capture user and the connect user.

- Capture Process Trace Files
  A capture process is an Oracle background process named `CP`*`nn`*, where *`nn`* can include letters and numbers.
- The XStream Out Client Application and Security
  There are security considerations for the client application because XStream Out allows it to receive LCRs.
- XStream Out Component-Level Security
  All the components of the XStream Out configuration run as the same XStream administrator. This user needs to be granted the `XSTREAM_CAPTURE` role.
- Privileges Required by the Capture User for a Capture Process
  Changes are captured in the security domain of the capture user for a capture process. The capture user captures all changes that satisfy the capture process rule sets. The capture user must have the necessary privileges to perform these actions.

- Privileges Required by the Connect User for an Outbound Server
  An outbound server sends LCRs to an XStream client application in the security domain of its connect user.

See Also:

- "XStream Security Model"
- *Oracle Database PL/SQL Packages and Types Reference*

Capture Process Trace Files

A capture process is an Oracle background process named `CP`*`nn`*, where *`nn`* can include letters and numbers.

For example, on some operating systems, if the system identifier for a database running a capture process is `hqdb` and the capture process number is `01`, then the trace file for the capture process starts with `hqdb_CP01`.

See Also:

"Displaying Change Capture Information About Each Capture Process" for a query that displays the capture process number of a capture process

The XStream Out Client Application and Security

There are security considerations for the client application because XStream Out allows it to receive LCRs.

After an XStream Out application receives LCRs, the application might save the contents of LCRs to a file or generate the SQL statements to execute the LCRs on a non-Oracle database.

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. XStream does not assume that the connected user to the outbound server is trusted.

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

See Also:

- *Oracle Call Interface Developer's Guide* for information about the OCI interface for XStream
- *Oracle Database XStream Java API Reference* for information about the Java interface for XStream

XStream Out Component-Level Security

All the components of the XStream Out configuration run as the same XStream administrator. This user needs to be granted the `XSTREAM_CAPTURE` role.

The `XSTREAM_CAPTURE` role contains privileges for Oracle for Oracle-supplied views and packages required to run components in an XStream Out configuration.

Privileges Required by the Capture User for a Capture Process

Changes are captured in the security domain of the capture user for a capture process. The capture user captures all changes that satisfy the capture process rule sets. The capture user must have the necessary privileges to perform these actions.

The capture user must have the following privileges:

- `EXECUTE` privilege on the rule sets used by the capture process
- `EXECUTE` privilege on all custom rule-based transformation functions specified for rules in the positive rule set
- Privileges to enqueue LCRs into the capture process queue

A capture process can be associated with only one user, but one user can be associated with many capture processes.

Grant privileges to the capture user with the `XSTREAM_CAPTURE` role.

See Also:

- "Configure an XStream Administrator on All Databases"
- "Changing the Capture User of an Outbound Server's Capture Process"

Privileges Required by the Connect User for an Outbound Server

An outbound server sends LCRs to an XStream client application in the security domain of its connect user.

The connect user sends LCRs that satisfy the outbound server's rule sets to the XStream client application. In addition, the connect user runs all custom rule-based transformations specified by the rules in these rule sets.

The connect user must have the following privileges:

- `EXECUTE` privilege on the rule sets used by the outbound server
- `EXECUTE` privilege on all custom rule-based transformation functions specified for rules in the positive rule set

A outbound server can be associated with only one user, but one user can be associated with many outbound servers.

See Also:

- "XStream Security Model"
- "Changing the Connect User for an Outbound Server"
- *Oracle Database PL/SQL Packages and Types Reference* `DBMS_XSTREAM_ADM` package "Security Model" for information about the security requirements for configuring and managing XStream
- *Oracle Call Interface Developer's Guide* for information about the OCI interface for XStream

XStream Out and Other Oracle Database Components

XStream Out can work with other Oracle Database components.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- XStream Out and Oracle Real Application Clusters
  XStream Out can work with Oracle Real Application Clusters (Oracle RAC).
- XStream Out and Transparent Data Encryption
  XStream Out can work with Transparent Data Encryption.
- XStream Out and Flashback Data Archive
  XStream Out supports tables in a flashback data archive.
- XStream Out and Recovery Manager
  RMAN deletion policies can affect capture processes.
- XStream and Distributed Transactions
  XStream Out supports distributed transactions.
- XStream Out and a Multitenant Environment
  A multitenant environment enables a database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.

XStream Out and Oracle Real Application Clusters

XStream Out can work with Oracle Real Application Clusters (Oracle RAC).

- Capture Processes and Oracle Real Application Clusters
  A capture process can capture changes in an Oracle Real Application Clusters (Oracle RAC) environment.
- Queues and Oracle Real Application Clusters
  You can configure queues in an Oracle Real Application Clusters (Oracle RAC) environment.
- Propagations and Oracle Real Application Clusters
  A propagation can propagate LCRs from one queue to another in an Oracle Real Application Clusters (Oracle RAC) environment. A propagation job running on an instance propagates logical change records (LCRs) from any queue owned by that instance to destination queues.
- Outbound Servers and Oracle Real Application Clusters
  You can configure an outbound server in an Oracle Real Application Clusters (Oracle RAC) environment provided you have set `use_rac_service` to `Y` in the procedure `DBMS_CAPTURE_ADM.SET_PARAMETER`.

Capture Processes and Oracle Real Application Clusters

A capture process can capture changes in an Oracle Real Application Clusters (Oracle RAC) environment.

If you use one or more capture processes and Oracle RAC in the same environment, then all archived logs that contain changes to be captured by a capture process must be available for all instances in the Oracle RAC environment. In an Oracle RAC environment, a capture process reads changes made by all instances. Multiple outbound server processes that use the same capture process must run in the same Oracle RAC instance as the capture process.

You ensure that the capture process runs in the same Oracle RAC instance as its queue by setting the parameter `use_rac_service` to `Y` in the procedure `DBMS_CAPTURE_ADM.SET_PARAMETER`.

If the value for the capture process parameter `use_rac_service` is set to `Y`, then each capture process is started and stopped on the owner instance for its `ANYDATA` queue, even if the start or stop procedure is run on a different instance. Also, a capture process follows its queue to a different instance if the current owner instance becomes unavailable. The queue itself follows the rules for primary instance and secondary instance ownership.

If the value for the capture process parameter `use_rac_service` is set to `N`, then the capture process is started on the instance to which the client application connects. Stopping the capture process must be performed on the same instance where the capture process was started.

If the owner instance for a queue table containing a queue used by a capture process becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster. In addition, if the capture process was enabled when the owner instance became unavailable, then the capture process is restarted automatically on the new owner instance. If the capture process was disabled when the owner instance became unavailable, then the capture process remains disabled on the new owner instance.

LogMiner supports the `LOG_ARCHIVE_DEST_`*`n`* initialization parameter, and capture processes use LogMiner to capture changes from the redo log. If an archived log file is inaccessible from one destination, then a local capture process can read it from another accessible destination. On an Oracle RAC database, this ability also enables you to use cross instance archival (CIA) such that each instance archives its files to all other instances. This solution cannot detect or resolve gaps caused by missing archived log files. Hence, it can be used only to complement an existing solution to have the archived files shared between all instances.

In a downstream capture process environment, the source database can be a single instance database or a multi-instance Oracle RAC database. The downstream database can be a single instance database or a multi-instance Oracle RAC database, regardless of whether the source database is single instance or multi-instance.

See Also:

- "Capture Processes"
- *Oracle Database Reference* for more information about the `ALL_QUEUE_TABLES` data dictionary view
- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about configuring archived logs to be shared between instances

Queues and Oracle Real Application Clusters

You can configure queues in an Oracle Real Application Clusters (Oracle RAC) environment.

In an Oracle RAC environment, only the owner instance can have a buffer for a queue, but different instances can have buffers for different queues. A buffered queue is System Global Area (SGA) memory associated with a queue.

You set the capture process parameter `use_rac_service` to `Y` to specify ownership of the queue table or the primary and secondary instance for a given queue table.

XStream Out processes and jobs support primary instance and secondary instance specifications for queue tables. If `use_rac_service` is set to `Y`, you can use the specifications for queue tables and the secondary instance assumes ownership of a queue table when the primary instance becomes unavailable. The queue ownership is transferred back to the primary instance when it becomes available again.

If the owner instance for a queue table containing a destination queue for a propagation becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster. If both the primary and secondary instance for a queue table containing a destination queue become unavailable, then queue ownership is transferred automatically to another instance in the cluster. In this case, if the primary or secondary instance becomes available again, then ownership is transferred back to one of them accordingly.

You can set primary and secondary instance specifications using the `ALTER_QUEUE_TABLE` procedure in the `DBMS_AQADM` package. The `ALL_QUEUE_TABLES` data dictionary view contains information about the owner instance for a queue table. A queue table can contain multiple queues. In this case, each queue in a queue table has the same owner instance as the queue table.

The `NETWORK_NAME` column in the `ALL_QUEUES` data dictionary view contains the network name for a queue service. Do not manage the services for queues in any way. Oracle manages them automatically.

See Also:

- *Oracle Database Reference* for more information about the `ALL_QUEUE_TABLES` data dictionary view
- *Oracle Database Advanced Queuing User's Guide* for more information about queues and Oracle RAC
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ALTER_QUEUE_TABLE` procedure

Propagations and Oracle Real Application Clusters

A propagation can propagate LCRs from one queue to another in an Oracle Real Application Clusters (Oracle RAC) environment. A propagation job running on an instance propagates logical change records (LCRs) from any queue owned by that instance to destination queues.

The information in this section only applies to XStream configurations that include propagations. In a typical XStream configuration, an outbound server and its capture process are configured on the same database, and propagation is not required. The information in this section does not apply to configurations that do not include propagation. However, it is possible to configure a capture process on one database and an outbound server on another database. In this case, a propagation sends LCRs from the capture process's queue to the outbound server's queue.

Before you can propagate LCRs in an Oracle RAC environment, you must set `use_rac_service` to `Y` in the procedure `DBMS_CAPTURE_ADM.SET_PARAMETER`.

Any propagation to an Oracle RAC database is made over database links. The database links must be configured to connect to the destination instance that owns the queue that will receive the LCRs.

A queue-to-queue propagation to a buffered destination queue uses a service to provide transparent failover in an Oracle RAC environment. That is, a propagation job for a queue-to-queue propagation automatically connects to the instance that owns the destination queue. The service used by a queue-to-queue propagation always runs on the owner instance of the destination queue. This service is created only for buffered queues in an Oracle RAC database. If you plan to use buffered messaging with an Oracle RAC database, then LCRs can be enqueued into a buffered queue on any instance. If LCRs are enqueued on an instance that does not own the queue, then the LCRs are sent to the correct instance, but it is more efficient to enqueue LCRs on the instance that owns the queue. You can use the service to connect to the owner instance of the queue before enqueuing LCRs into a buffered queue.

Because the queue service always runs on the owner instance of the queue, transparent failover can occur when Oracle RAC instances fail. When multiple queue-to-queue propagations use a single database link, the connect description for each queue-to-queue propagation changes automatically to propagate LCRs to the correct destination queue.

Note:

If a queue contains or will contain captured LCRs in an Oracle RAC environment, then use queue-to-queue propagations to propagate LCRs to an Oracle RAC destination database.

Outbound Servers and Oracle Real Application Clusters

You can configure an outbound server in an Oracle Real Application Clusters (Oracle RAC) environment provided you have set `use_rac_service` to `Y` in the procedure `DBMS_CAPTURE_ADM.SET_PARAMETER`.

Each outbound server is started and stopped on the owner instance for its `ANYDATA` queue, even if the start or stop procedure is run on a different instance. A coordinator process, its corresponding apply reader server, and its apply server run on a single instance. Multiple XStream Out processes that use the same capture process must run in the same Oracle RAC instance as the capture process.

If the owner instance for a queue table containing a queue used by an outbound server becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster. Also, an outbound server will follow its queue to a different instance if the current owner instance becomes unavailable. The queue itself follows the rules for primary instance and secondary instance ownership. In addition, if the outbound server was enabled when the owner instance became unavailable, then the outbound server is restarted automatically on the new owner instance. If the outbound server was disabled when the owner instance became unavailable, then the outbound server remains disabled on the new owner instance.

See Also:

- "Outbound Servers"
- *Oracle Database Reference* for more information about the `ALL_QUEUE_TABLES` data dictionary view

XStream Out and Transparent Data Encryption

XStream Out can work with Transparent Data Encryption.

- Capture Processes and Transparent Data Encryption
  Capture processes can capture changes to columns that have been encrypted using Transparent Data Encryption.
- Outbound Servers and Transparent Data Encryption
  An outbound server can process implicitly captured row logical change records (row LCRs) that contain columns encrypted using Transparent Data Encryption.

See Also:

Oracle Database Advanced Security Guide for information about Transparent Data Encryption

Capture Processes and Transparent Data Encryption

Capture processes can capture changes to columns that have been encrypted using Transparent Data Encryption.

A local capture process can capture changes to columns that have been encrypted using Transparent Data Encryption. A downstream capture process can capture changes to columns that have been encrypted only if the downstream database shares an encryption keystore (container for authentication and signing credentials) with the source database. A keystore can be shared through a network file system (NFS), or it can be copied from one computer system to another manually. When a keystore is shared with a downstream database, ensure that the `WALLET_ROOT` parameter in the `sqlnet.ora` file at the downstream database specifies the keystore location.

If you copy a keystore to a downstream database, then ensure that you copy the keystore from the source database to the downstream database whenever the keystore at the source database changes. Do not perform any operations on the keystore at the downstream database, such as changing the encryption key for a replicated table.

Encrypted columns in row logical change records (row LCRs) captured by a local or downstream capture process are decrypted when the row LCRs are staged in a buffered queue.

Note:

A capture process only supports encrypted columns if the redo logs used by the capture process were generated by a database with a compatibility level of 11.0.0 or higher. The compatibility level is controlled by the `COMPATIBLE` initialization parameter.

Note:

The `SQLNET.ENCRYPTION_WALLET_LOCATION sqlnet.ora` parameter is deprecated in Oracle Database 19c.

Starting with Oracle Database 23ai, the parameter `ENCRYPTION_WALLET_LOCATION` is desupported.

See Also:

"Capture Processes"

Outbound Servers and Transparent Data Encryption

An outbound server can process implicitly captured row logical change records (row LCRs) that contain columns encrypted using Transparent Data Encryption.

When row LCRs with encrypted columns are processed by an outbound server, the encrypted columns are decrypted. These row LCRs with decrypted columns are sent to the XStream client application.

When row LCRs with encrypted columns are stored in buffered queues, the columns are decrypted. When row LCRs spill to disk, XStream transparently encrypts any encrypted columns while the row LCRs are stored on disk.

Note:

For XStream Out to encrypt columns transparently, the encryption master key must be stored in the keystore on the local database, and the keystore must be open. The following statements set the master key and open the keystore:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY key-password;

ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY key-password;
```

Because the same keystore needs to be available and open in any instance where columns are encrypted, make sure you copy the keystore to the downstream capture database. In the case of a downstream capture, you must also run the above commands on the downstream instance.

See Also:

"Outbound Servers"

XStream Out and Flashback Data Archive

XStream Out supports tables in a flashback data archive.

Capture processes can capture data manipulation language (DML) and data definition language (DDL) changes made to these tables. Outbound servers can process the captured LCRs.

XStream Out also support the following DDL statements:

- `CREATE FLASHBACK ARCHIVE`
- `ALTER FLASHBACK ARCHIVE`
- `DROP FLASHBACK ARCHIVE`
- `CREATE TABLE` with a `FLASHBACK ARCHIVE` clause
- `ALTER TABLE` with a `FLASHBACK ARCHIVE` clause

Note:

XStream Out does not capture changes made to internal tables used by a flashback data archive.

See Also:

- *Oracle Database Development Guide* for information about flashback data archive
- "Capture Processes"
- "Outbound Servers"

XStream Out and Recovery Manager

RMAN deletion policies can affect capture processes.

Some RMAN deletion policies and commands delete archived redo log files. If one of these RMAN policies or commands is used on a database that generates redo log files for one or more capture processes, then ensure that the RMAN commands do not delete archived redo log files that are required by a capture process.

- RMAN and Local Capture Processes
  When a local capture process is configured, RMAN does not delete archived redo log files that are required by the local capture process unless there is space pressure in the fast recovery area.
- RMAN and Downstream Capture Processes
  When a downstream capture process captures database changes made at a source database, ensure that no RMAN deletion policy or command deletes an archived redo log file until after it is transferred from the source database to the downstream capture process database.

See Also:

- "Capture Processes"
- "The Capture Process Is Missing Required Redo Log Files" for information about determining whether a capture process is missing required archived redo log files and for information correcting this problem
- "Checking the Trace File and Alert Log for Problems"
- *Oracle Database Backup and Recovery User's Guide* and *Oracle Database Backup and Recovery Reference* for more information about RMAN

RMAN and Local Capture Processes

When a local capture process is configured, RMAN does not delete archived redo log files that are required by the local capture process unless there is space pressure in the fast recovery area.

Specifically, RMAN does not delete archived redo log files that contain changes with system change number (SCN) values that are equal to or greater than the required checkpoint SCN for the local capture process. This is the default RMAN behavior for all RMAN deletion policies and `DELETE` commands, including `DELETE ARCHIVELOG` and `DELETE OBSOLETE`.

When there is not enough space in the fast recovery area to write a new log file, RMAN automatically deletes one or more archived redo log files. Oracle Database writes warnings to the alert log when RMAN automatically deletes an archived redo log file that is required by a local capture process.

When backups of the archived redo log files are taken on the local capture process database, Oracle recommends the following RMAN deletion policy:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO BACKED UP integer TIMES
   TO DEVICE TYPE deviceSpecifier;
```

This deletion policy requires that a log file be backed up *`integer`* times before it is considered for deletion.

When no backups of the archived redo log files are taken on the local capture process database, no specific deletion policy is recommended. By default, RMAN does not delete archived redo log files that are required by a local capture process.

RMAN and Downstream Capture Processes

When a downstream capture process captures database changes made at a source database, ensure that no RMAN deletion policy or command deletes an archived redo log file until after it is transferred from the source database to the downstream capture process database.

The following are considerations for specific RMAN deletion policies and commands that delete archived redo log files:

- The RMAN command `CONFIGURE ARCHIVELOG DELETION POLICY` sets a deletion policy that determines when archived redo log files in the fast recovery area are eligible for deletion. The deletion policy also applies to all RMAN `DELETE` commands, including `DELETE ARCHIVELOG` and `DELETE OBSOLETE`.

  The following settings determine the behavior at the source database:

  - A deletion policy set `TO SHIPPED TO STANDBY` does not delete a log file until after it is transferred to a downstream capture process database that requires the file. These log files might or might not have been processed by the downstream capture process. Automatic deletion occurs when there is not enough space in the fast recovery area to write a new log file.
  - A deletion policy set `TO APPLIED ON STANDBY` does not delete a log file until after it is transferred to a downstream capture process database that requires the file and the source database marks the log file as applied. The source database marks a log file as applied when the minimum required checkpoint SCN of all of the downstream capture processes for the source database is greater than the highest SCN in the log file.
  - A deletion policy set to `BACKED UP` *`integer`* `TIMES TO DEVICE TYPE` requires that a log file be backed up *`integer`* times before it is considered for deletion. A log file can be deleted even if the log file has not been processed by a downstream capture process that requires it.
  - A deletion policy set `TO NONE` means that a log file can be deleted when there is space pressure on the fast recovery area, even if the log file has not been processed by a downstream capture process that requires it.
- The RMAN command `DELETE ARCHIVELOG` deletes archived redo log files that meet all of the following conditions:
  - The log files satisfy the condition specified in the `DELETE ARCHIVELOG` command.

  - The log files can be deleted according to the `CONFIGURE ARCHIVELOG DELETION POLICY`. For example, if the policy is set `TO SHIPPED TO STANDBY`, then this command does not delete a log file until after it is transferred to any downstream capture process database that requires it.

  This behavior applies when the database is mounted or open.

  If archived redo log files are not deleted because they contain changes required by a downstream capture process, then RMAN displays a warning message about skipping the delete operation for these files.

- The RMAN command `DELETE OBSOLETE` permanently purges the archived redo log files that meet all of the following conditions:
  - The log files are obsolete according to the retention policy.
  - The log files can be deleted according to the `CONFIGURE ARCHIVELOG DELETION POLICY`. For example, if the policy is set `TO SHIPPED TO STANDBY`, then this command does not delete a log file until after it is transferred to any downstream capture process database that requires it.

  This behavior applies when the database is mounted or open.
- The RMAN command `BACKUP ARCHIVELOG ALL DELETE INPUT` copies the archived redo log files and deletes the original files after completing the backup. This command does not delete the log file until after it is transferred to a downstream capture process database when the following conditions are met:
  - The database is mounted or open.
  - The log file is required by a downstream capture process.
  - The deletion policy is set `TO SHIPPED TO STANDBY`.

  If archived redo log files are not deleted because they contain changes required by a downstream capture process, then RMAN displays a warning message about skipping the delete operation for these files.

Oracle recommends one of the following RMAN deletion policies at the source database for a downstream capture process:

- When backups of the archived redo log files are taken on the source database, set the deletion policy to the following:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO SHIPPED TO STANDBY
   BACKED UP integer TIMES TO DEVICE TYPE deviceSpecifier;
```

- When no backups of the archived redo log files are taken on the source database, set the deletion policy to the following:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO SHIPPED TO STANDBY;
```

Note:

At a downstream capture process database, archived redo log files transferred from a source database are not managed by RMAN.

XStream and Distributed Transactions

XStream Out supports 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.

A capture process captures changes made to the source database during a distributed transaction using either of these two methods and sends the changes to an outbound server. An outbound server sends the changes in a transaction to a client application after the transaction has committed.

However, the distributed transaction state is not sent. The client application does not inherit the in-doubt or prepared state of such a transaction. Also, the outbound server does not 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 Out supports changes made by loosely coupled XA branches regardless of the `COMPATIBLE` initialization parameter value. XStream Out supports replication of changes made by tightly coupled branches on an Oracle RAC source database only if the `COMPATIBLE` initialization parameter set to `11.2.0.0` or higher.

See Also:

- *Oracle Database Administrator's Guide* for more information about distributed transactions
- *Oracle Database Development Guide* for more information about Oracle XA

XStream Out and a Multitenant Environment

A multitenant environment enables a database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.

This self-contained collection is called a pluggable database (PDB). A multitenant container database (CDB) contains PDBs. In a CDB, XStream Out functions much the same as it does in a non-CDB.

A CDB can also contain application containers. An application container is an optional component of a CDB that consists of an application root and all application PDBs associated with it. An application container stores data for one or more applications. An application container shares application metadata and common data. In a CDB, each of the following is a container: the CDB root, each PDB, each application root, and each application PDB.

The main differences in the way XStream Out functions in a CDB and non-CDB are:

- XStream Out must be configured only in the CDB root.
- XStream Out can see changes made to any container within the CDB.

- XStream Out capture rules can limit the LCRs to those that are needed for the client application. The system-generated capture rules select the appropriate LCRs based on the parameters that were passed to the `ADD_OUTBOUND` and `CREATE_OUTBOUND` procedures in the `DBMS_XSTREAM_ADM` package. You can use the `ADD_*_RULES` procedures in the same package for more fine-grained control over the rules used by the XStream Out components.
- The user who performs XStream Out tasks must be a common user.

Unplug and Plug Operations in an XStream Environment

When a PDB, application root, or application PDB involved with XStream Out is unplugged from its CDB and plugged into another CDB, any capture process or outbound server is not considered part of the container. You must configure the capture process and outbound server again in the other CDB.

If an outbound server is configured in a different database than the capture process, then unplug and plug operations have additional considerations.

For this example, assume the following:

- A CDB named `CDB1` contains PDB `PDB1`.
- A capture process is configured in `CDB1`, and it sends LCRs from `PDB1` to an outbound server in a CDB named `CDB2`.
- You unplug `PDB1` from `CDB1`, and then plug it into a CDB named `CDB3`.

To continue delivering LCRs from `PDB1` to the outbound server in `CDB2`, you must configure a new capture process in `CDB3` to capture and send LCRs to `CDB2`.

The rules used by the outbound server in database B must be altered to change references to the root of `CDB1` to the root of `CDB3`. In addition, if `PDB1` was given a different name in `CDB3`, then the rules must be altered to reflect the new PDB name.

Application Containers in an XStream Environment

When a CDB has one or more application containers, XStream Out must be configured in the CDB root, and XStream Out can capture changes made in any container in the CDB, including the application roots and application PDBs. Changes captured in an application container can be sent to containers of any type, including PDBs, application roots, and application PDBs.

When replicating changes from one application root to another application root, XStream can replicate `ALTER PLUGGABLE DATABASE APPLICATION` statements. To avoid errors, the target application root that applies the statements must have the same application installed as the source application root, and the application name must be identical in both application roots.

To avoid errors when replicating changes from an application root to a container that is not an application root, you must ensure that `ALTER PLUGGABLE DATABASE APPLICATION` statements are not replicated.

With the XStream OCI API, you can control whether `ALTER PLUGGABLE DATABASE APPLICATION` statements are replicated using the `OCIXStreamOutAttach` function and the `OCILCRHeaderGet` function. With the XStream Java API, you can control this behavior using the `mode` parameter in the `XStreamOut.attach` method.

- Configure a Multitenant Container Database

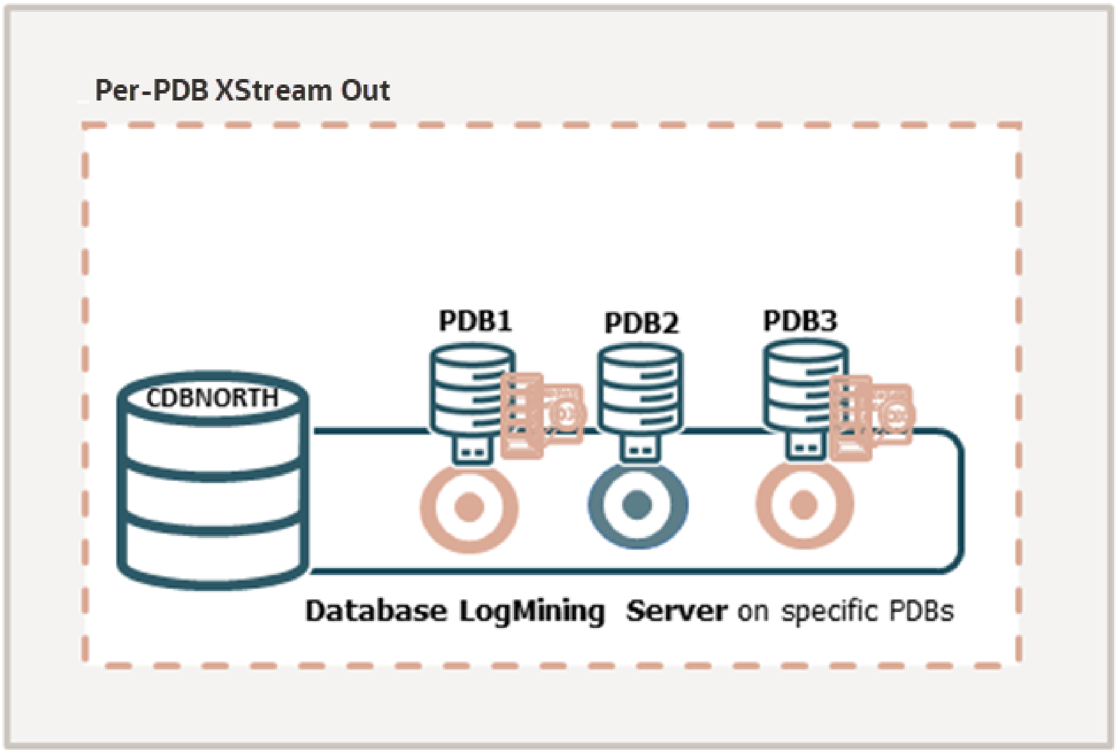
Related Topics

- System-Created Rules and a Multitenant Environment
  A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.
- Configuring XStream Out in a CDB
  When you configure XStream Out in a CDB, you must decide which database changes will be captured by XStream Out and sent to the client application.
- *Oracle Multitenant Administrator's Guide*

Configure a Multitenant Container Database

Oracle Database 23ai and higher releases allow each pluggable database (PDB) can have XStream Out registered for a specific PDB, which is called per-PDB XStream Out.

The following diagram shows the multitenant container database configuration:



Adding XStream Out directly from the PDB. This approach is useful when XStream Out captures from isolated PDBs, managing ownership and responsibility at the PDB level.

Using a per-PDB XStream Out, you can connect as the local PDB user (for example, `xstrmadmin`) and then register this XStream Out with the database. As you are already logged in as the PDB user, an additional `container` clause is *not* required. Similarly, the `SOURCECATALOG` or a three-part naming convention is also not needed.

Considerations for Multitenant Container Database Configuration

Consider the following guidelines when configuring a multitenant container databases for data replication using Oracle Database XStream:

- The different pluggable databases in the multitenant container database can have different character sets. Oracle Database XStream captures data from any multitenant database with different character sets into one trail file and replicates the data without corruption due to using different character sets.
- To create and register a per-PDB XStream Out, you will need to connect to the PDB user such as `xstrmadmin` created for PDB-level access.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. Starting with Oracle Database 23ai, only Per-PDB XStream Out is supported. While the documentation is being revised, legacy terminology may persist. In most cases, "database", "non-CDB", and "CDB" refer to PDB level actions only. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

4 Configuring XStream Out

You can configure the Oracle Database components that are used by XStream Out.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- Preparing for XStream Out
  There are decisions to make and tasks to complete to prepare for an XStream Out configuration.
- Configuring XStream Out
  An outbound server in an XStream Out configuration streams Oracle database changes to a client application.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- *Oracle Call Interface Developer's Guide*
- *Oracle Database XStream Java API Reference*

Preparing for XStream Out

There are decisions to make and tasks to complete to prepare for an XStream Out configuration.

- Decide How to Configure XStream Out
  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).
- Prerequisites for Configuring XStream Out
  Preparing for an XStream Out outbound server is similar to preparing for an Oracle Replication environment.

Decide How to Configure XStream Out

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. The source database is the database where the changes were generated. Downstream capture means that a capture process runs on a database other than the source database. 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.
- **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 4-2 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

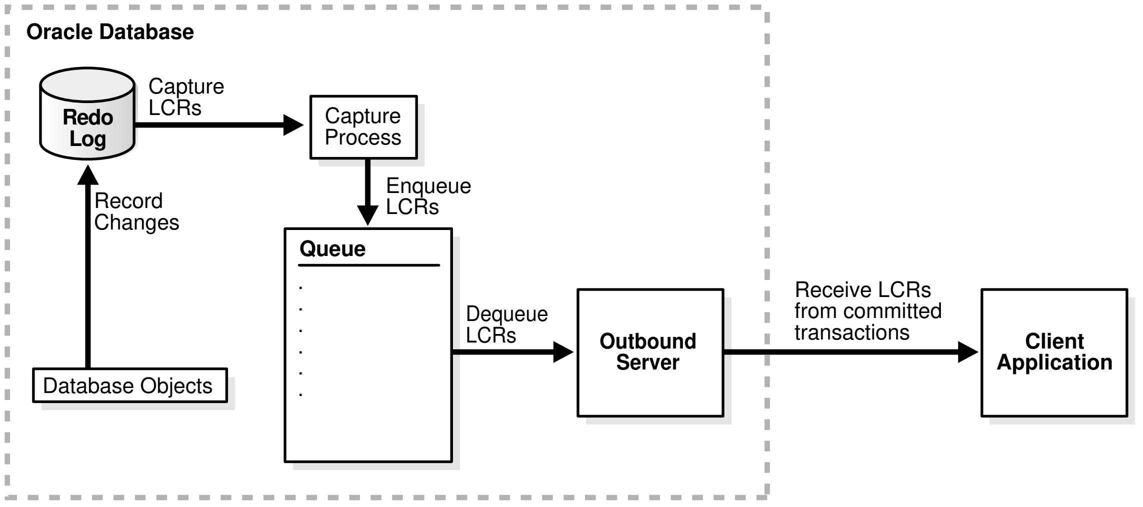
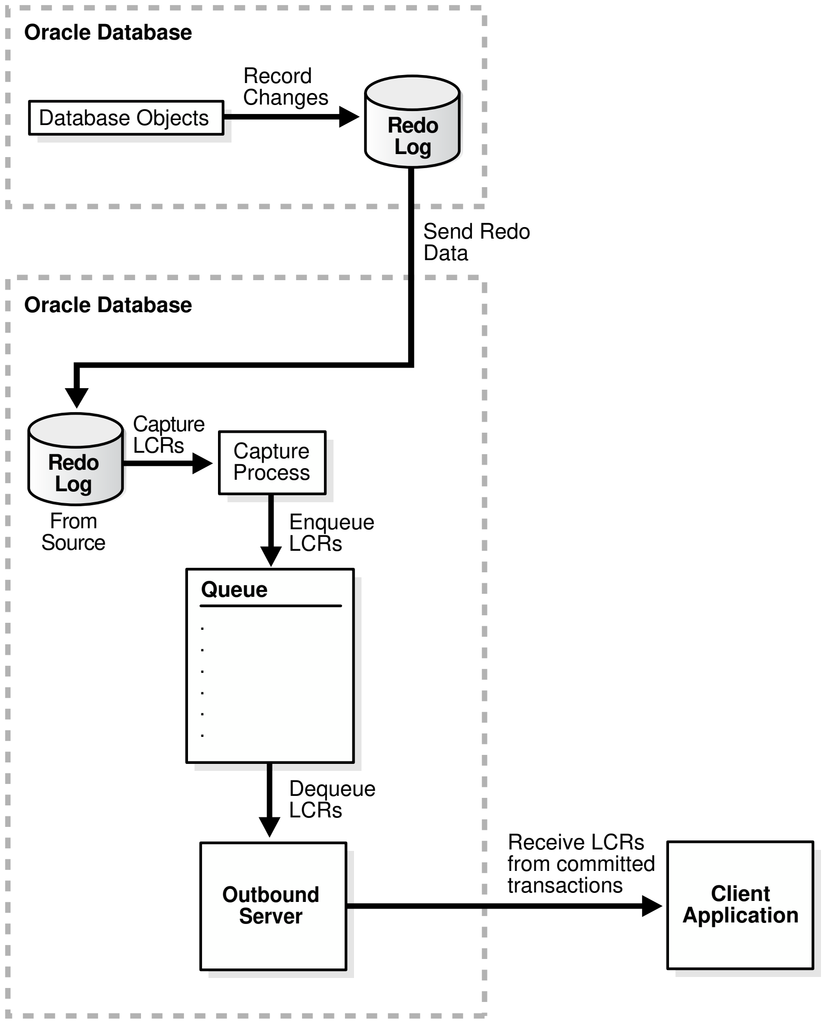


Figure 4-2 Downstream Capture and Outbound Server in the Same Database



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 at the source database sends 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, where the capture process captures changes in real-time.
- 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:

"Local Capture and Downstream Capture"

Prerequisites for Configuring XStream Out

Preparing for an XStream Out outbound server is similar to preparing for an Oracle Replication environment.

The components used in an Oracle 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.

This section provides an overview of each task and specific information about completing the task for an XStream Out configuration.

- Configure an XStream Administrator on All Databases
  An XStream administrator configures and manages XStream components in an XStream Out environment.
- Granting Additional Privileges to the XStream Administrator
  Additional privileges might be required for the XStream administrator.
- Grant User Privileges for Oracle Database 23ai and Higher

- If Required, Configure Network Connectivity and Database Links
  Network connectivity and database links are required when an XStream Out configuration includes multiple databases.
- 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.
- 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.
- Configure the Streams pool
  The Streams pool is a portion of memory in the System Global Area (SGA) that is used by bothOracle Streams and XStream components.
- If Required, Configure Supplemental Logging
  When you use a capture process to capture changes, supplemental logging must be specified for certain columns at a source database for changes to the columns to be applied successfully at a destination database.
- 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.
- 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.

Configure an XStream Administrator on All Databases

An XStream administrator configures and manages XStream components in an XStream Out environment.

You can configure 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 user name of the XStream administrator is `xstrmadmin` for a non-CDB. The user name of the XStream administrator is `c##xstrmadmin` for a multitenant container database (CDB).
- 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.

   If you are configuring an XStream administrator for XStream Out in a CDB, then connect to the root and configure the XStream administrator in the CDB root.

   See Also:

   - *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus
   - "XStream Out and a Multitenant Environment" for more information about XStream Out and CDBs

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.

   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;
   ```

   If you are creating an XStream administrator in a CDB, then you must create the tablespace in all of the containers in the CDB, including the CDB root, all pluggable databases (PDBs), all application roots, and all application containers. The tablespace is required in all containers because the XStream administrator must be a common user and so must have access to the tablespace in any container.

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;
   ```

   If you are creating an XStream administrator in a CDB, then the XStream administrator must be a common user. Therefore, include the `CONTAINER=ALL` clause in the `CREATE USER` statement:

   ```
   CREATE USER c##xstrmadmin IDENTIFIED BY password
     DEFAULT TABLESPACE xstream_tbs
     QUOTA UNLIMITED ON xstream_tbs
     CONTAINER=ALL;
   ```

Note:

Enter an appropriate password for the administrative user.

See Also:

Oracle Database Security Guide for guidelines about choosing passwords

4. Grant `CREATE SESSION` privilege to the XStream administrator.

   If you created a new user to act as the XStream administrator, then grant this user `CREATE SESSION` privilege.

   For example, to grant `CREATE SESSION` privilege to user `xstrmadmin`, run the following statement:

   ```
   GRANT CREATE SESSION TO xstrmadmin;
   ```

   If you are creating an XStream administrator in a CDB, then grant `CREATE SESSION` privilege and `SET CONTAINER` privilege to the XStream administrator, and include the `CONTAINER=ALL` clause in the statement.

   For example, to grant these privileges to user `c##xstrmadmin` in a CDB, run the following statement:

   ```
   GRANT CREATE SESSION, SET CONTAINER TO c##xstrmadmin CONTAINER=ALL;
   ```

5. Grant the `XSTREAM_CAPTURE` role to the XStream administrator.

   See Also:

   Oracle Database PL/SQL Packages and Types Reference

6. If necessary, grant additional privileges to the XStream administrator.

   See "Granting Additional Privileges to the XStream Administrator".

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

Example 4-1 Granting Privileges to a XStream Administrator in a Non-CDB

```
GRANT XSTREAM_CAPTURE TO xstrmadmin;
```

Granting Additional Privileges to the XStream Administrator

Additional privileges might be required for the XStream administrator.

Grant any of the following additional privileges to the XStream administrator if necessary:

- If you plan to use Oracle Enterprise Manager Cloud Control to manage databases with XStream components, then the XStream administrator must be granted `DBA` role. You must also configure the XStream administrator to be an Oracle Enterprise Manager administrative user. Doing so grants additional privileges required by Oracle Enterprise

Manager Cloud Control, such as the privileges required to run Oracle Enterprise Manager Cloud Control jobs. See the Oracle Enterprise Manager Cloud Control online help for information about creating Oracle Enterprise Manager administrative users.

- Grant the privileges for a remote XStream administrator to perform actions in the local database. 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 is 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.
- 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, propagation, or outbound server. For a capture process, if a capture user is specified, then the capture 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, propagation, or outbound 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 LCRs into or dequeue LCRs from the queue. The XStream administrator might also need `ENQUEUE` or `DEQUEUE` privileges on the queue, or both.
- 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 you are using Oracle Database Vault, then the XStream administrator must be granted the `DV_XSTREAM_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_XSTREAM_ADMIN` role from the XStream administrator.

  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

  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*.

Grant User Privileges for Oracle Database 23ai and Higher

Oracle Database 23ai uses a role-based approach to grant privileges necessary for replication. The roles are a replacement of the `GRANT_ADMIN_PRIVILEGE` procedures from the `DBMS_XSTREAM_AUTH` package.

For example, to create an Oracle XStream Administration user, you have to grant the appropriate `XSTREAM_CAPTURE` or `XSTREAM_APPLY` roles to the user while creating the user. If you still use `GRANT_ADMIN_PRIVILEGE` procedures from the `DBMS_XSTREAM_AUTH` package in Oracle Database 23ai, no privileges are granted. Instead, a warning message is raised alerting that the procedure call is disabled.

Here are the user roles introduced in Oracle XStream 23ai:

`XSTREAM_CAPTURE`
`XSTREAM_CAPTURE` has the privileges necessary for using and managing XStream Out processes.
For capturing DML and DDL with Oracle XStream Out, a user requires the following permissions:

```
GRANT CONNECT, RESOURCE to xstrmadmin;
GRANT XSTREAM_CAPTURE to xstrmadmin;
```

A downstream XStream Out process is registered at the root container level. When configuring a downstream XStream Out process, the following step is required:

```
ALTER USER c##xstrmadmin CONTAINER_DATA = all CONTAINER = current;
```

`XSTREAM_APPLY`
Role with privileges necessary for using Oracle XStream In.
To use Oracle XStream In processes a user needs this role, as well as the permissions to execute DML and DDL at the target. For example, if an Oracle XStream In process is intended to perform DML operations on the `EMPLOYEES` table from the `HR` schema:

```
GRANT CONNECT, RESOURCE to xstrmadmin;
GRANT XSTRM_APPLY to xstrmadmin;
GRANT SELECT, INSERT, UPDATE, DELETE on HR.EMPLOYEES;
```

If the user is intended to apply DDL operations like `CREATE TABLE`, `DROP TABLE` and `ALTER TABLE` it should receive the system privileges necessary to execute such statements:

```
GRANT CREATE TABLE, ALTER TABLE, DROP TABLE to xstrmadmin;
```

`XSTRM_APPLY_PROCREP`
Role with privileges necessary to execute packages supported for procedural replication with Oracle XStream. It only includes the execution permissions, therefore, this role should be used together with `XSTREAM_APPLY` role to allow the user to run the Oracle XStream In process and to execute the procedures at the target.

For example, if an Oracle XStream In will apply procedure executions. The grant process should be as follows:

```
GRANT CONNECT, RESOURCE to xstrmadmin;
GRANT XSTRM_APPLY, XSTRM_APPLY_PROCREP to xstrmadmin;
```

If Required, Configure Network Connectivity and Database Links

Network connectivity and database links are required when an XStream Out configuration includes multiple databases.

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 Out" 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 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.

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 Streams pool

The Streams pool is a portion of memory in the System Global Area (SGA) that is used by bothOracle Streams and XStream components.

The Streams pool stores buffered queue LCRs in memory, and it provides memory for capture processes and outbound servers.

The following are considerations for configuring the Streams pool:

- Oracle recommends that you set the MAX_SGA_SIZE parameter to 1GB as a starting point. The Streams Pool Utilization might be higher or lower. Depending on the workload, you can increase or decrease the Streams_Pool_Size. If there are multiple Outbound or Inbound servers, then Oracle recommends that you use the SUM of any MAX_SGA_SIZES with a 25 percent offset.
- After XStream Out is configured, you can use the `max_sga_size` capture process parameter to control the amount of system global area (SGA) memory allocated specifically to a capture process.

  The sum of system global area (SGA) memory allocated for all components on a database must be less than the value set for the `STREAMS_POOL_SIZE` initialization parameter.
- After XStream Out is configured, you can use the `max_sga_size` apply parameter to control the amount of SGA memory allocated specifically to an outbound server.
- Ensure that there is enough space in the Streams pool at each database to run XStream components and to store LCRs and run the components properly.
- The Streams pool is initialized the first time an outbound server is started.
- The best practice is to set the `STREAMS_POOL_SIZE` initialization parameter explicitly to the desired Streams pool size.

The Automatic Shared Memory Management feature automatically manages the size of the Streams pool when the following conditions are met:

- The `MEMORY_TARGET` and `MEMORY_MAX_TARGET` initialization parameters are both set to `0` (zero).
- The `SGA_TARGET` initialization parameter is set to a nonzero value.

The Streams pool size is the value specified by the `STREAMS_POOL_SIZE` parameter, in bytes, if the following conditions are met:

- The `MEMORY_TARGET`, `MEMORY_MAX_TARGET`, and `SGA_TARGET` initialization parameters are all set to `0` (zero).
- The `STREAMS_POOL_SIZE` initialization parameter is set to a nonzero value.

If you are using Automatic Shared Memory Management, and if the `STREAMS_POOL_SIZE` initialization parameter also is set to a nonzero value, then Automatic Shared Memory Management uses this value as a minimum for the Oracle Streams pool. If your environment needs a minimum amount of memory in the Oracle Streams pool to function properly, then you can set a minimum size. To view the current memory allocated to Oracle Streams pool by Automatic Shared Memory Management, query the `V$SGA_DYNAMIC_COMPONENTS` view. In addition, you can query the `V$STREAMS_POOL_STATISTICS` view to view the current usage of the Oracle Streams pool.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `max_sga_size` capture process parameter
- *Oracle Database Administrator’s Guide*
- *Oracle Database Reference*

If Required, Configure Supplemental Logging

When you use a capture process to capture changes, supplemental logging must be specified for certain columns at a source database for changes to the columns to be applied successfully at a destination database.

Supplemental logging places additional information in the redo log for these columns. A capture process captures this additional information and places it in logical change records (LCRs), and an XStream inbound server or client application might need this additional information to process changes properly.

- Required Supplemental Logging in an XStream Environment
  There are two types of supplemental logging: database supplemental logging and table supplemental logging.
- Specifying Table Supplemental Logging Using Unconditional Log Groups
  You can specify supplemental logging using unconditional log groups.
- Specifying Table Supplemental Logging Using Conditional Log Groups
  You can specify table supplemental logging using conditional log groups.
- Dropping a Supplemental Log Group
  To drop a conditional or unconditional supplemental log group, use the `DROP SUPPLEMENTAL LOG GROUP` clause in the `ALTER TABLE` statement.
- Specifying Database Supplemental Logging of Key Columns
  You have the option of specifying supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database.
- Dropping Database Supplemental Logging of Key Columns
  To drop supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database, issue the `ALTER DATABASE DROP SUPPLEMENTAL LOG DATA` statement.
- Procedures That Automatically Specify Supplemental Logging
  Some procedures in the `DBMS_CAPTURE_ADM` package automatically specify supplemental logging.

Required Supplemental Logging in an XStream Environment

There are two types of supplemental logging: database supplemental logging and table supplemental logging.

Database supplemental logging specifies supplemental logging for an entire database, while table supplemental logging enables you to specify log groups for supplemental logging of a particular table. If you use table supplemental logging, then you can choose between two types of log groups: unconditional log groups and conditional log groups.

Unconditional log groups log the before images of specified columns when the table is changed, regardless of whether the change affected any of the specified columns. Unconditional log groups are sometimes referred to as "always log groups." Conditional log groups log the before images of all specified columns only if at least one of the columns in the log group is changed.

Supplementing logging at the database level, unconditional log groups at the table level, and conditional log groups at the table level determine which old values are logged for a change.

If you plan to use one or more XStream inbound servers to apply LCRs captured by a capture process, then you must enable supplemental logging *at the source database* for the following types of columns in tables *at the destination database*:

- Any columns at the source database that are used in a primary key in tables for which changes are applied at a destination database must be unconditionally logged in a log group or by database supplemental logging of primary key columns.
- If the parallelism of any inbound server that will apply the changes is greater than 1, then any unique constraint column at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging does not need to be specified if a unique constraint column comes from a single column at the source database.
- If the parallelism of any inbound server that will apply the changes is greater than 1, then any foreign key column at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging does not need to be specified if the foreign key column comes from a single column at the source database.
- If the parallelism of any inbound server that will apply the changes is greater than 1, then any bitmap index column at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging does not need to be specified if the bitmap index column comes from a single column at the source database.
- Any columns at the source database that are used as substitute key columns for an inbound server at a destination database must be unconditionally logged. You specify substitute key columns for a table using the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package.
- The columns specified in a column list for conflict resolution during apply must be conditionally logged if multiple columns at the source database are used in the column list at the destination database.
- Any columns at the source database that are used by a change handler, procedure DML handler, or error handler at a destination database must be unconditionally logged.
- Any columns at the source database that are used by a rule or a rule-based transformation must be unconditionally logged.
- Any columns at the source database that are specified in a value dependency virtual dependency definition at a destination database must be unconditionally logged.
- If you specify row subsetting for a table at a destination database, then any columns at the source database that are in the destination table or columns at the source database that are in the subset condition must be unconditionally logged. You specify a row subsetting condition for an inbound server using the `dml_condition` parameter in the `ADD_SUBSET_RULES` procedure in the `DBMS_XSTREAM_ADM` package.

If you do not use supplemental logging for these types of columns at a source database, then changes involving these columns might not apply properly at a destination database.

Note:

Columns of the following data types cannot be part of a supplemental log group: LOB, `LONG`, `LONG RAW`, user-defined types (including object types, `REF`s, varrays, nested tables), and Oracle-supplied types (including `Any` types, XML types, spatial types, and media types).

Specifying Table Supplemental Logging Using Unconditional Log Groups

You can specify supplemental logging using unconditional log groups.

To specify an unconditional supplemental log group that only includes the primary key column(s) for a table, use an `ALTER TABLE` statement with the `PRIMARY KEY` option in the `ADD SUPPLEMENTAL LOG DATA` clause. For example, the following statement adds the primary key column of the `hr.regions` table to an unconditional log group with a system-generated name:

```
ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

To specify an unconditional supplemental log group that includes all of the columns in a table, use an `ALTER TABLE` statement with the `ALL` option in the `ADD SUPPLEMENTAL LOG DATA` clause. For example, the following statement adds all of the columns in the `hr.regions` table to an unconditional log group with a system-generated name:

```
ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

To specify an unconditional supplemental log group that contains columns that you select, use an `ALTER TABLE` statement with the `ALWAYS` specification for the `ADD SUPPLEMENTAL LOG GROUP` clause. These log groups can include key columns, if necessary.

For example, the following statement adds the `department_id` column and the `manager_id` column of the `hr.departments` table to an unconditional log group named `log_group_dep_pk`:

```
ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_dep_pk
  (department_id, manager_id) ALWAYS;
```

The `ALWAYS` specification makes this log group an unconditional log group.

Specifying Table Supplemental Logging Using Conditional Log Groups

You can specify table supplemental logging using conditional log groups.

You can use the following options in the `ADD SUPPLEMENTAL LOG DATA` clause of an `ALTER TABLE` statement:

- The `FOREIGN KEY` option creates a conditional log group that includes the foreign key column(s) in the table.
- The `UNIQUE` option creates a conditional log group that includes the unique key column(s) and bitmap index column(s) in the table.

If you specify multiple options in a single `ALTER TABLE` statement, then a separate conditional log group is created for each option.

For example, the following statement creates two conditional log groups:

```
ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG DATA
  (UNIQUE, FOREIGN KEY) COLUMNS;
```

One conditional log group includes the unique key columns and bitmap index columns for the table, and the other conditional log group includes the foreign key columns for the table. Both log groups have a system-generated name.

Note:

Specifying the UNIQUE option does not enable supplemental logging of bitmap join index columns.

To specify a conditional supplemental log group that includes any columns you choose to add, you can use the ADD SUPPLEMENTAL LOG GROUP clause in the ALTER TABLE statement. To make the log group conditional, do not include the ALWAYS specification.

For example, suppose the min_salary and max_salary columns in the hr.jobs table are included in a column list for conflict resolution at a destination database. The following statement adds the min_salary and max_salary columns to a conditional log group named log_group_jobs_cr:

```
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_cr
  (min_salary, max_salary);
```

Dropping a Supplemental Log Group

To drop a conditional or unconditional supplemental log group, use the DROP SUPPLEMENTAL LOG GROUP clause in the ALTER TABLE statement.

For example, to drop a supplemental log group named log_group_jobs_cr, run the following statement:

```
ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_cr;
```

Specifying Database Supplemental Logging of Key Columns

You have the option of specifying supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database.

You might choose this option if you configure a capture process to capture changes to an entire database. To specify supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database, issue the following SQL statement:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
   (PRIMARY KEY, UNIQUE, FOREIGN KEY) COLUMNS;
```

If your primary key, unique key, bitmap index, and foreign key columns are the same at all source and destination databases, then running this command at the source database provides the supplemental logging needed for primary key, unique key, bitmap index, and foreign key columns at all destination databases. When you specify the PRIMARY KEY option, all columns of a row's primary key are placed in the redo log file any time the table is modified (unconditional logging). When you specify the UNIQUE option, any columns in a row's unique key and bitmap index are placed in the redo log file if any column belonging to the unique key or bitmap index is modified (conditional logging). When you specify the FOREIGN KEY option, all columns of a row's foreign key are placed in the redo log file if any column belonging to the foreign key is modified (conditional logging).

You can omit one or more of these options. For example, if you do not want to supplementally log all of the foreign key columns in the database, then you can omit the FOREIGN KEY option, as in the following example:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
   (PRIMARY KEY, UNIQUE) COLUMNS;
```

In addition to PRIMARY KEY, UNIQUE, and FOREIGN KEY, you can also use the ALL option. The ALL option specifies that, when a row is changed, all the columns of that row (except for LOB, LONG, LONG RAW, user-defined type, and Oracle-supplied type columns) are placed in the redo log file (unconditional logging).

Supplemental logging statements are cumulative. If you issue two consecutive ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statements, each with a different identification key, then both keys are supplementally logged.

Note:

Specifying the UNIQUE option does not enable supplemental logging of bitmap join index columns.

See Also:

Oracle Database SQL Language Reference for information about data types

Dropping Database Supplemental Logging of Key Columns

To drop supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database, issue the ALTER DATABASE DROP SUPPLEMENTAL LOG DATA statement.

For example, to drop database supplemental logging for all primary key, unique key, bitmap index, and foreign key columns, issue the following SQL statement:

```
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA
  (PRIMARY KEY, UNIQUE, FOREIGN KEY) COLUMNS;
```

Note:

Dropping database supplemental logging of key columns does not affect any existing table-level supplemental log groups.

Procedures That Automatically Specify Supplemental Logging

Some procedures in the DBMS_CAPTURE_ADM package automatically specify supplemental logging.

The following procedures in the DBMS_CAPTURE_ADM package automatically specify supplemental logging:

- BUILD
- PREPARE_GLOBAL_INSTANTIATION
- PREPARE_SCHEMA_INSTANTIATION
- PREPARE_TABLE_INSTANTIATION

The BUILD procedure automatically specifies database supplemental logging by running the ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statement. In most cases, the BUILD procedure is run automatically when a capture process is created.

The PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures automatically specify supplemental logging of the primary key, unique key, bitmap index, and foreign key columns in the tables prepared for instantiation.

Certain procedures in the DBMS_XSTREAM_ADM package automatically run a procedure listed previously, including the ADD_SUBSET_RULES, ADD_TABLE_RULES, ADD_SCHEMA_RULES, and ADD_GLOBAL_RULES procedures.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about these procedures

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 Out" for information about this decision.

Tip:

You can use Oracle Enterprise Manager Cloud Control to configure log file transfer and a downstream capture process. See the Oracle Enterprise Manager Cloud Control online help for instructions.

The steps in this section configure the source database to transfer its redo log files to the capture database and configure the capture database to accept these redo log files.

To configure log file transfer to a downstream database:

1. Configure Oracle Net so that the source database can communicate with the downstream database.

   See Also:

   Oracle Database Net Services Administrator's Guide

2. Configure authentication at both databases to support the transfer of redo data.

   Redo transport sessions are authenticated using either the Secure Sockets Layer (SSL) protocol or a remote login password file. If the source database has a remote login password file, then copy it to the appropriate directory on the downstream capture database system. The password file must be the same at the source database and the downstream capture database.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for detailed information about authentication requirements for redo transport

3. At the source database, set the following initialization parameters to configure redo transport services to transmit redo data from the source database to the downstream database:

   - `LOG_ARCHIVE_DEST_n` - Configure at least one `LOG_ARCHIVE_DEST_n` initialization parameter to transmit redo data to the downstream database. Set the following attributes of this parameter in the following way:

     – `SERVICE` - Specify the network service name of the downstream database.

     – `ASYNC` or `SYNC` - Specify a redo transport mode.

       The advantage of specifying `ASYNC` is that it results in little or no effect on the performance of the source database. `ASYNC` is recommended to avoid affecting source database performance if the downstream database or network is performing poorly.

       The advantage of specifying `SYNC` is that redo data is sent to the downstream database faster than when `ASYNC` is specified. Also, specifying `SYNC AFFIRM` results in behavior that is similar to `MAXIMUM AVAILABILITY` standby protection mode. Note that specifying an `ALTER DATABASE STANDBY DATABASE TO MAXIMIZE AVAILABILITY` SQL statement has no effect on an XStream capture process.

     – `NOREGISTER` - Specify this attribute so that the location of the archived redo log files is not recorded in the downstream database control file.

     – `VALID_FOR` - Specify either `(ONLINE_LOGFILE,PRIMARY_ROLE)` or `(ONLINE_LOGFILE,ALL_ROLES)`.

     – `TEMPLATE` - If you are configuring an archived-log downstream capture process, then specify a directory and format template for archived redo logs at the downstream database. The `TEMPLATE` attribute overrides the `LOG_ARCHIVE_FORMAT` initialization parameter settings at the downstream database. The `TEMPLATE` attribute is valid only with remote destinations. Ensure that the format uses all of the following variables at each source database: `%t`, `%s`, and `%r`.

       Do not specify the `TEMPLATE` attribute if you are configuring a real-time downstream capture process.

     – `DB_UNIQUE_NAME` - The unique name of the downstream database. Use the name specified for the `DB_UNIQUE_NAME` initialization parameter at the downstream database.

     The following example is a `LOG_ARCHIVE_DEST_n` setting that specifies the downstream database `dbs2` for a real-time downstream capture process:

     ```
     LOG_ARCHIVE_DEST_2='SERVICE=DBS2.EXAMPLE.COM ASYNC NOREGISTER
        VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
        DB_UNIQUE_NAME=dbs2'
     ```

The following example is a `LOG_ARCHIVE_DEST_n` setting that specifies the downstream database `dbs2` for an archived-log downstream capture process:

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.EXAMPLE.COM ASYNC NOREGISTER
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
   TEMPLATE=/usr/oracle/log_for_dbs1/dbs1_arch_%t_%s_%r.log
   DB_UNIQUE_NAME=dbs2'
```

See "Decide How to Configure XStream Out" for information about the differences between real-time and archived-log downstream capture.

Tip:

If you are configuring an archived-log downstream capture process, then specify a value for the `TEMPLATE` attribute that keeps log files from a remote source database separate from local database log files. In addition, if the downstream database contains log files from multiple source databases, then the log files from each source database should be kept separate from each other.

- `LOG_ARCHIVE_DEST_STATE_n` - Set this initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_n` parameter for the downstream database to `ENABLE`.

  For example, if the `LOG_ARCHIVE_DEST_2` initialization parameter is set for the downstream database, then set the `LOG_ARCHIVE_DEST_STATE_2` parameter in the following way:

  ```
  LOG_ARCHIVE_DEST_STATE_2=ENABLE
  ```

- `LOG_ARCHIVE_CONFIG` - Set the `DB_CONFIG` attribute in this initialization parameter to include the `DB_UNIQUE_NAME` of the source database and the downstream database.

  For example, if the `DB_UNIQUE_NAME` of the source database is `dbs1`, and the `DB_UNIQUE_NAME` of the downstream database is `dbs2`, then specify the following parameter:

  ```
  LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbs1,dbs2)'
  ```

  By default, the `LOG_ARCHIVE_CONFIG` parameter enables a database to both send and receive redo.

See Also:

Oracle Database Reference and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

4. At the downstream database, set the `DB_CONFIG` attribute in the `LOG_ARCHIVE_CONFIG` initialization parameter to include the `DB_UNIQUE_NAME` of the source database and the downstream database.

   For example, if the `DB_UNIQUE_NAME` of the source database is `dbs1`, and the `DB_UNIQUE_NAME` of the downstream database is `dbs2`, then specify the following parameter:

   ```
   LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbs1,dbs2)'
   ```

By default, the `LOG_ARCHIVE_CONFIG` parameter enables a database to both send and receive redo.

5. If you reset any initialization parameters while the instance was running at a database in Step 3 or Step 4, then you might want to reset them in the initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while the instance was running, but instead reset them in the initialization parameter file in Step 3 or Step 4, then restart the database. The source database must be open when it sends redo log files to the downstream database, because the global name of the source database is sent to the downstream database only if the source database is open.

When these steps are complete, you can add standby redo logs files at the downstream database if you want to configure a real-time downstream capture process. In this case, see the instructions in "If Required, Add Standby Redo Logs for Real-Time Downstream Capture".

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 Out" for information about this decision.

The example in this section adds standby redo logs at a downstream database. Standby redo logs are required to configure a real-time downstream capture process. In the example, the source database is `dbs1.example.com` and the downstream database is `dbs2.example.com`

The steps in this section are required only if you are configuring real-time downstream capture. If you are configuring archived-log downstream capture, then do not complete the steps in this section.

To add standby redo logs for real-time downstream capture:

1. Complete the steps in "If Required, Configure Log File Transfer to a Downstream Database".
2. At the downstream database, set the following initialization parameters to configure archiving of the redo data generated locally:
   - Set at least one archive log destination in the `LOG_ARCHIVE_DEST_`*`n`* initialization parameter either to a directory or to the fast recovery area on the computer system running the downstream database. Set the following attributes of this parameter in the following way:
     – `LOCATION` - Specify either a valid path name for a disk directory or, to use a fast recovery area, specify `USE_DB_RECOVERY_FILE_DEST`. This location is the local destination for archived redo log files written from the standby redo logs. Log files from a remote source database should be kept separate from local database log files. See *Oracle Database Backup and Recovery User’s Guide* for information about configuring a fast recovery area.
     – `VALID_FOR` - Specify either `(ONLINE_LOGFILE,PRIMARY_ROLE)` or `(ONLINE_LOGFILE,ALL_ROLES)`.

     The following example is a `LOG_ARCHIVE_DEST_`*`n`* setting for the locally generated redo data at the real-time downstream capture database:

     ```
     LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local_rl_dbs2
        VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)'
     ```

A real-time downstream capture configuration should keep archived standby redo log files separate from archived online redo log files generated by the downstream database. Specify `ONLINE_LOGFILE` instead of `ALL_LOGFILES` for the redo log type in the `VALID_FOR` attribute to accomplish this.

You can specify other attributes in the `LOG_ARCHIVE_DEST_`*`n`* initialization parameter if necessary.

- Set the `LOG_ARCHIVE_DEST_STATE_`*`n`* initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_`*`n`* parameter previously set in this step to `ENABLE`.

  For example, if the `LOG_ARCHIVE_DEST_1` initialization parameter is set, then set the `LOG_ARCHIVE_DEST_STATE_1` parameter in the following way:

  ```
  LOG_ARCHIVE_DEST_STATE_1=ENABLE
  ```

3. At the downstream database, set the following initialization parameters to configure the downstream database to receive redo data from the source database and write the redo data to the standby redo log at the downstream database:

   - Set at least one archive log destination in the `LOG_ARCHIVE_DEST_`*`n`* initialization parameter either to a directory or to the fast recovery area on the computer system running the downstream database. Set the following attributes of this parameter in the following way:

     – `LOCATION` - Specify either a valid path name for a disk directory or, to use a fast recovery area, specify `USE_DB_RECOVERY_FILE_DEST`. This location is the local destination for archived redo log files written from the standby redo logs. Log files from a remote source database should be kept separate from local database log files. See *Oracle Database Backup and Recovery User's Guide* for information about configuring a fast recovery area.

     – `VALID_FOR` - Specify either `(STANDBY_LOGFILE,PRIMARY_ROLE)` or `(STANDBY_LOGFILE,ALL_ROLES)`.

     The following example is a `LOG_ARCHIVE_DEST_`*`n`* setting for the redo data received from the source database at the real-time downstream capture database:

     ```
     LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbs1
        VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
     ```

     You can specify other attributes in the `LOG_ARCHIVE_DEST_`*`n`* initialization parameter if necessary.

   - Set the `LOG_ARCHIVE_DEST_STATE_`*`n`* initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_`*`n`* parameter previously set in this step to `ENABLE`.

     For example, if the `LOG_ARCHIVE_DEST_2` initialization parameter is set for the downstream database, then set the `LOG_ARCHIVE_DEST_STATE_2` parameter in the following way:

     ```
     LOG_ARCHIVE_DEST_STATE_2=ENABLE
     ```

See Also:

*Oracle Database Reference*and*Oracle Data Guard Concepts and Administration*for more information about these initialization parameters

4. If you reset any initialization parameters while an instance was running at a database in Step 2 or Step 3, then you might want to reset them in the relevant initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while an instance was running, but instead reset them in the initialization parameter file in Step 2 or Step 3, then restart the database. The source database must be open when it sends redo data to the downstream database, because the global name of the source database is sent to the downstream database only if the source database is open.

5. Create the standby redo log files.

   Note:

   The following steps outline the general procedure for adding standby redo log files to the downstream database. The specific steps and SQL statements used to add standby redo log files depend on your environment. For example, in an Oracle Real Application Clusters (Oracle RAC) environment, the steps are different. See *Oracle Data Guard Concepts and Administration* for detailed instructions about adding standby redo log files to a database.

   a. In SQL*Plus, connect to the source database `dbs1.example.com` as an administrative user.

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

   b. Determine the log file size used on the source database. The standby log file size must exactly match (or be larger than) the source database log file size. For example, if the source database log file size is 500 MB, then the standby log file size must be 500 MB or larger. You can determine the size of the redo log files at the source database (in bytes) by querying the `V$LOG` view at the source database.

      For example, query the `V$LOG` view:

      ```
      SELECT BYTES FROM V$LOG;
      ```

   c. Determine the number of standby log file groups required on the downstream database.

      The number of standby log file groups must be at least one more than the number of online log file groups on the source database. For example, if the source database has two online log file groups, then the downstream database must have at least three standby log file groups.

      You can determine the number of source database online log file groups by querying the `V$LOG` view of the source database for a single instance database or by querying the `GV$LOG` view for a database cluster.

      For example, query the `GV$LOG` view:

      ```
      SELECT COUNT(GROUP#) FROM GV$LOG;
      ```

   d. In SQL*Plus, connect to the downstream database `dbs2.example.com` as an administrative user.

   e. Use the SQL statement `ALTER DATABASE ADD STANDBY LOGFILE` to add the standby log file groups to the downstream database.

For example, assume that the source database has two online redo log file groups and is using a log file size of 500 MB. In this case, use the following statements to create the appropriate standby log file groups:

```
ALTER DATABASE ADD STANDBY LOGFILE GROUP 3
   ('/oracle/dbs/slog3a.rdo', '/oracle/dbs/slog3b.rdo') SIZE 500M;

ALTER DATABASE ADD STANDBY LOGFILE GROUP 4
   ('/oracle/dbs/slog4.rdo', '/oracle/dbs/slog4b.rdo') SIZE 500M;

ALTER DATABASE ADD STANDBY LOGFILE GROUP 5
   ('/oracle/dbs/slog5.rdo', '/oracle/dbs/slog5b.rdo') SIZE 500M;
```

f. Ensure that the standby log file groups were added successfully by running the following query:

```
SELECT GROUP#, THREAD#, SEQUENCE#, ARCHIVED, STATUS
   FROM V$STANDBY_LOG;
```

You output should be similar to the following:

```
    GROUP#    THREAD#  SEQUENCE# ARC STATUS
---------- ---------- ---------- --- ----------
         3          0          0 YES UNASSIGNED
         4          0          0 YES UNASSIGNED
         5          0          0 YES UNASSIGNED
```

g. Ensure that log files from the source database are appearing in the location specified in the `LOCATION` attribute in Step 3. You might need to switch the log file at the source database to see files in the directory.

When these steps are complete, you are ready to configure a real-time downstream capture process.

Tip:

You can use Oracle Enterprise Manager Cloud Control to configure real-time downstream capture. See the Oracle Enterprise Manager Cloud Control online help for instructions.

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.

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 can create an outbound server, or it can add an outbound server to an existing XStream Out configuration. When you use this procedure on a database without an existing XStream Out configuration, it 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.

In addition, there are special considerations when you are configuring XStream Out in a CDB. This section provides instructions for creating outbound servers in a CDB.

Tip:

In an XStream Out configuration with multiple outbound servers, the best practice is to create one capture process that captures changes for all of the outbound servers.

- Configuring an Outbound Server 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.
- 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. You can add an additional outbound server to a database that already includes at least one outbound server.
- Configuring an Outbound Server Using ADD_OUTBOUND
  The ADD_OUTBOUND procedure in the DBMS_XSTREAM_ADM package creates an outbound server.
- Configuring XStream Out in a CDB
  When you configure XStream Out in a CDB, you must decide which database changes will be captured by XStream Out and sent to the client application.

Configuring an Outbound Server 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".

Assumptions

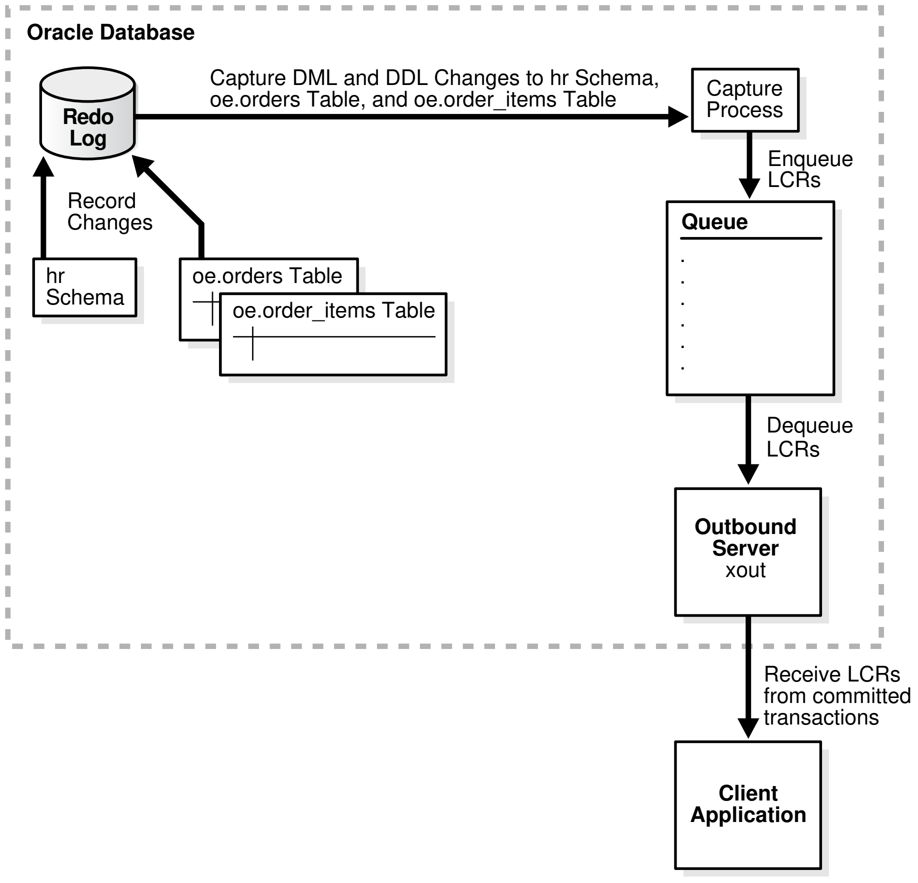
This section makes the following assumptions:

- 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 Out".
- 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 4-3 provides an overview of this XStream Out configuration.

Figure 4-3 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.

   Note:

   The `server_name` value cannot exceed 30 bytes.

   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 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".

When you run the client application, the outbound server is started automatically.

See Also:

Oracle Database PL/SQL Packages and Types Reference

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. You can add 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 an Outbound Server Using CREATE_OUTBOUND"
- "Configuring an Outbound Server Using ADD_OUTBOUND"

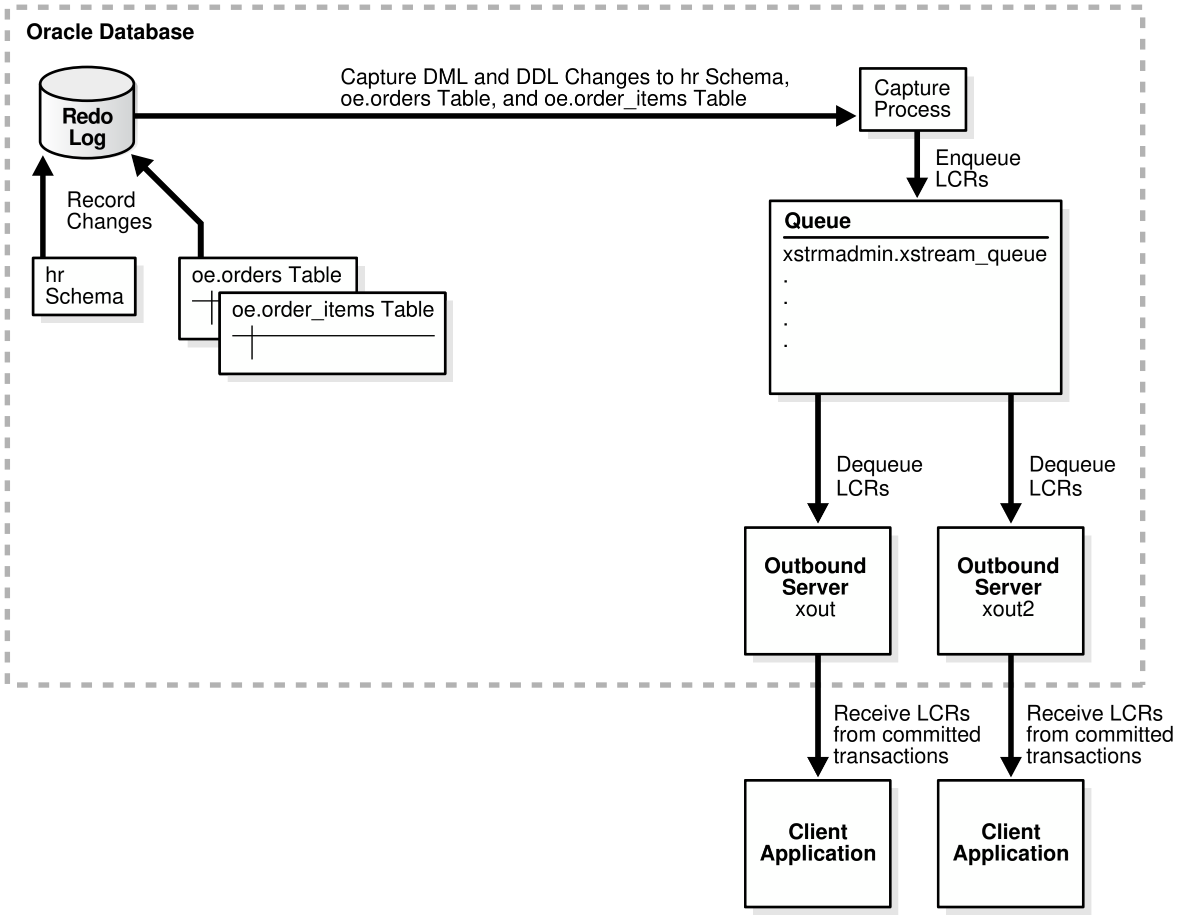
Assumptions

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 4-4 provides an overview of this XStream Out configuration.

Figure 4-4   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" 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.

Note:

The `server_name` value cannot exceed 30 bytes.

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 for a sample application.

When you run the client application, the outbound server is started automatically.

See Also:

Oracle Database PL/SQL Packages and Types Reference

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. In a database without an existing XStream Out configuration, you must configure these components manually.

You can use the `ADD_OUTBOUND` procedure to set up any of the configurations described in "Decide How to Configure XStream Out". However, if you chose to configure local capture and outbound server on the same database, then it is usually easier to use the `CREATE_OUTBOUND` procedure to configure all of the components simultaneously. See "Configuring an Outbound Server Using CREATE_OUTBOUND".

This section includes an example that configures downstream capture and the outbound server in the same database.

Prerequisites

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

- Complete the tasks described in "Prerequisites for Configuring XStream Out".

  If you decide to use downstream capture, then you must configure log file transfer from the source database to a downstream database. See "If Required, Configure Log File Transfer to a Downstream Database".

  If you want to use real-time downstream capture, then you must also add the required standby redo logs. See "If Required, Add Standby Redo Logs for Real-Time Downstream Capture".

  The example in this section uses downstream capture. Therefore, log file transfer must be configured to complete the example.

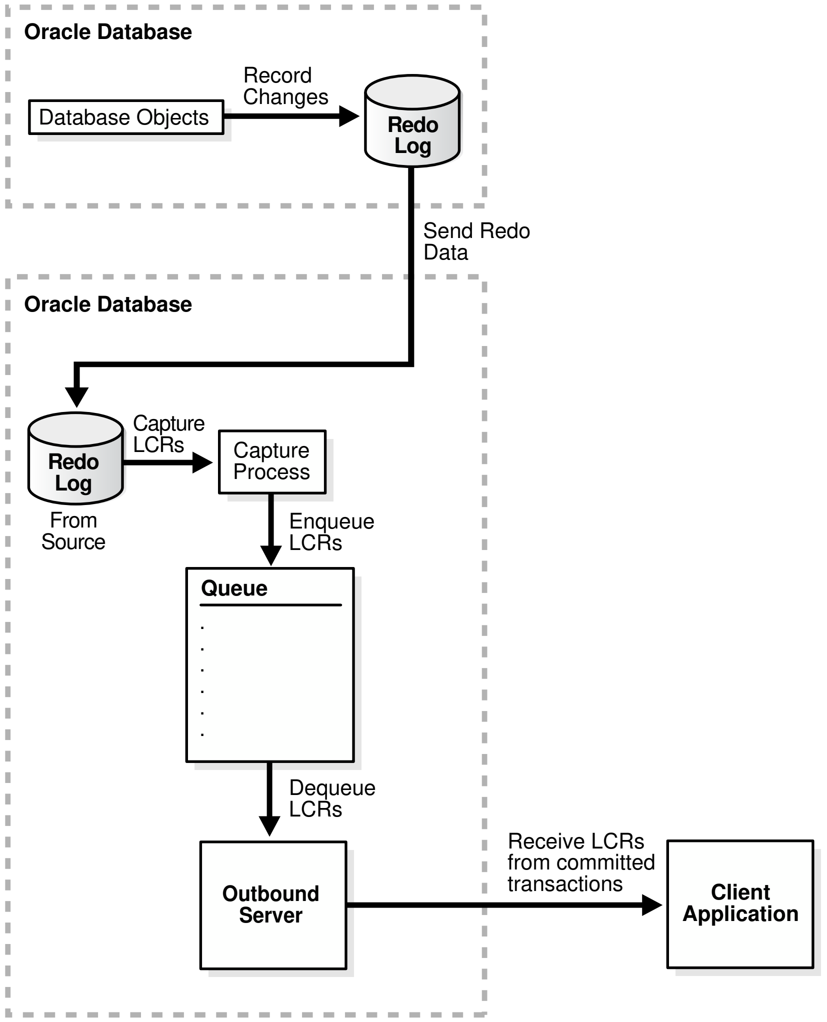
Assumptions

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`.
- The capture process and outbound server run on a different database than the source database. Therefore, downstream capture is configured.
- 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.

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

Figure 4-5 Sample XStream Out Configuration Created Using ADD_OUTBOUND



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.

   For example, run the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.SET_UP_QUEUE(
    queue_table => 'xstrmadmin.xstream_queue_table',
    queue_name  => 'xstrmadmin.xstream_queue');
END;
/
```

3. Create the database link from the downstream capture database to the source database.

   In this example, create a database link from the downstream capture database to `db1.example.com`. For example, if the user `xstrmadmin` is the XStream administrator on both databases, then create the following database link:

```
CREATE DATABASE LINK db1.example.com CONNECT TO xstrmadmin
   IDENTIFIED BY password USING 'db1.example.com';
```

See "If Required, Configure Network Connectivity and Database Links".

If you do not create the database link, then you must complete the following steps in source database:

a. Connect to the source database as the XStream administrator.

b. Run the `DBMS_CAPTURE_ADM.BUILD` procedure. For example:

```
SET SERVEROUTPUT ON
DECLARE
  scn  NUMBER;
BEGIN
  DBMS_CAPTURE_ADM.BUILD(
    first_scn => scn);
  DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
END;
/
First SCN Value = 409391
```

This procedure displays the valid first SCN value for the capture process that will be created in the downstream capture database. Make a note of the SCN value returned because you will use it when you create the capture process in Step 4.

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

For this example, ensure that supplemental logging is configured for the `hr` schema, the `oe.orders` table, and the `oe.order_items` table in the `db1.example.com` database.

See "If Required, Configure Supplemental Logging" for instructions about specifying supplemental logging.

These steps are not required if you create the database link.

4. While connected to the downstream capture database, create the capture process and add rules to it.

For example, run the following procedure to create the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'xstrmadmin.xstream_queue',
    capture_name       => 'xout_capture',
    capture_class      => 'xstream');
END;
/
```

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.

For example, run the following procedures to create the rules:

```
BEGIN
  DBMS_XSTREAM_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'capture',
    streams_name    => 'xout_capture',
    queue_name      => 'xstrmadmin.xstream_queue',
    include_dml     => TRUE,
    include_ddl     => TRUE,
    source_database => 'db1.example.com');
```

```
END;
/

BEGIN
  DBMS_XSTREAM_ADM.ADD_TABLE_RULES(
    table_name      =>  'oe.orders',
    streams_type    => 'capture',
    streams_name    => 'xout_capture',
    queue_name      => 'xstrmadmin.xstream_queue',
    include_dml     => TRUE,
    include_ddl     => TRUE,
    source_database => 'db1.example.com');
END;
/

BEGIN
  DBMS_XSTREAM_ADM.ADD_TABLE_RULES(
    table_name      =>  'oe.order_items',
    streams_type    => 'capture',
    streams_name    => 'xout_capture',
    queue_name      => 'xstrmadmin.xstream_queue',
    include_dml     => TRUE,
    include_ddl     => TRUE,
    source_database => 'db1.example.com');
END;
/
```

Do not start the capture process.

5. 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;
/
```

   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.

Note:

The `server_name` value cannot exceed 30 bytes.

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.

6. Create and run the client application that will connect to the outbound server and receive the LCRs. See Sample XStream Client Application for a sample application.

   When you run the client application, the outbound server is started automatically.

7. To add one or more additional outbound servers that receive LCRs from the capture process created in Step 4, follow the instructions in "Adding an Additional Outbound Server to a Capture Process Stream".

See Also:

Oracle Database PL/SQL Packages and Types Reference

Configuring XStream Out in a CDB

When you configure XStream Out in a CDB, you must decide which database changes will be captured by XStream Out and sent to the client application.

XStream Out can stream all database changes for all containers, including the CDB root and all PDBs, application roots, and application PDBs, or XStream Out can stream the changes from specific containers. In addition, you can configure XStream Out with local capture, or you can configure it with downstream capture to offload the work required to capture changes from the source database.

The following restrictions apply when you configure XStream Out in a CDB:

- The capture process and outbound server must be in the CDB root.
- The capture process and outbound server must be in the same CDB.
- Each container in the CDB must be open during XStream Out configuration.
- When changes made to an application root are captured, you must ensure that `ALTER PLUGGABLE DATABASE APPLICATION` statements are replicated only to other application roots.

In addition, ensure that you create the XStream administrator properly for a CDB.

Note:

When a container is created using a non-CDB, any XStream Out components from the non-CDB cannot be used in the container. You must drop and re-create the XStream Out components, including the capture process and outbound servers, in the CDB root.

- Configuring XStream Out with Local Capture in a CDB
  An example illustrates configuring XStream Out with local capture in a CDB.
- Configuring XStream Out with Downstream Capture in CDBs
  Using downstream capture, the XStream Out components can reside in databases other than the source database.

Related Topics

- XStream Out and a Multitenant Environment
  A multitenant environment enables a database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.
- System-Created Rules in a CDB and XStream Out
  In a CDB, XStream Out must be configured in the CDB root. Therefore, the PL/SQL procedures in the `DBMS_XSTREAM_ADM` package that create system-created rules must be run in the CDB root while connected as a common user.
- Configure an XStream Administrator on All Databases
  An XStream administrator configures and manages XStream components in an XStream Out environment.
- *Oracle Multitenant Administrator's Guide*

Configuring XStream Out with Local Capture in a CDB

An example illustrates configuring XStream Out with local capture in a CDB.

Prerequisites

Before configuring XStream Out, ensure that all containers in the CDB are in open read/write mode during XStream Out configuration.

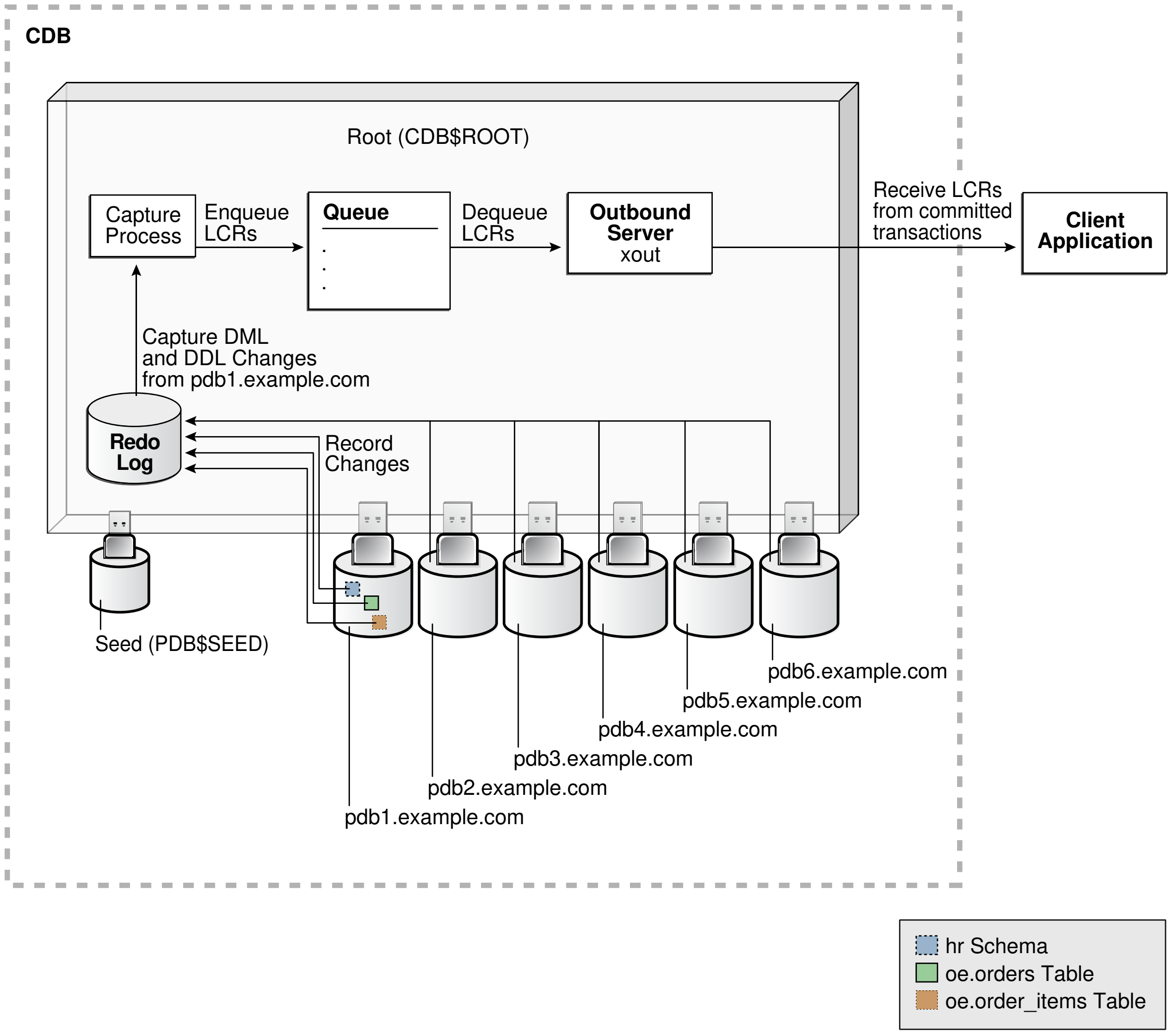
Assumptions

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 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 in PDB `pdb1.example.com` are sent to the outbound server.
- DML and DDL changes made to the `hr` schema in the PDB `pdb1.example.com` are sent to the outbound server.

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

Figure 4-6 Sample XStream Out Configuration Created Using CREATE_OUTBOUND for a PDB



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

1. In SQL*Plus, connect to the root in the CDB (not to the PDB `pdb1.example.com`) as the XStream administrator.
2. Create the outbound server and other XStream components.
   a. Ensure that all containers in the source CDB are in open read/write mode.
   b. Run the `CREATE_OUTBOUND` procedure.

      Given the assumptions for this example, 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',
    source_database =>  'pdb1.example.com',
    table_names     =>  tables,
    schema_names    =>  schemas);
END;
/
```

Note:

To capture changes in all containers in a CDB, including the CDB root, all PDBs, all application roots, and all application PDBs, and send those changes to the XStream client application, you can omit the `source_database` parameter when you run the `CREATE_OUTBOUND` procedure.

c. After the `CREATE_OUTBOUND` procedure completes successfully, optionally change the open mode of one or more containers if necessary.

Running the procedure in Step b performs the following actions:

- Configures supplemental logging for the `oe.orders` and `oe.order_items` tables and for all tables in the `hr` schema in the `pdb1.example.com` PDB.
- 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 from the `pdb1.example.com` PDB.
- 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.

Note:

The `server_name` value cannot exceed 30 bytes.

Tip:

To capture and send all database changes from the `pdb1.example.com` database 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 in the root of the CDB and receive the LCRs.

   When you run the client application, the outbound server is started automatically.

Related Topics

- Prerequisites for Configuring XStream Out
  Preparing for an XStream Out outbound server is similar to preparing for an Oracle Replication environment.
- Sample XStream Client Application
  Examples illustrate how to configure the Oracle Database components that are used by XStream. The examples configure sample client applications that communicate with an XStream outbound server and inbound server.

Configuring XStream Out with Downstream Capture in CDBs

Using downstream capture, the XStream Out components can reside in databases other than the source database.

When you have multiple CDBs, the source database can be in one CDB, and you can use downstream capture to capture the changes in another CDB.

Prerequisites

Before configuring XStream Out, the following prerequisites must be met:

- Ensure that all containers in the CDB are in open read/write mode during XStream Out configuration.
- This example uses downstream capture. Therefore, you must configure log file transfer from the source database to a downstream database.
- If you want to use real-time downstream capture, then you must also add the required standby redo logs.

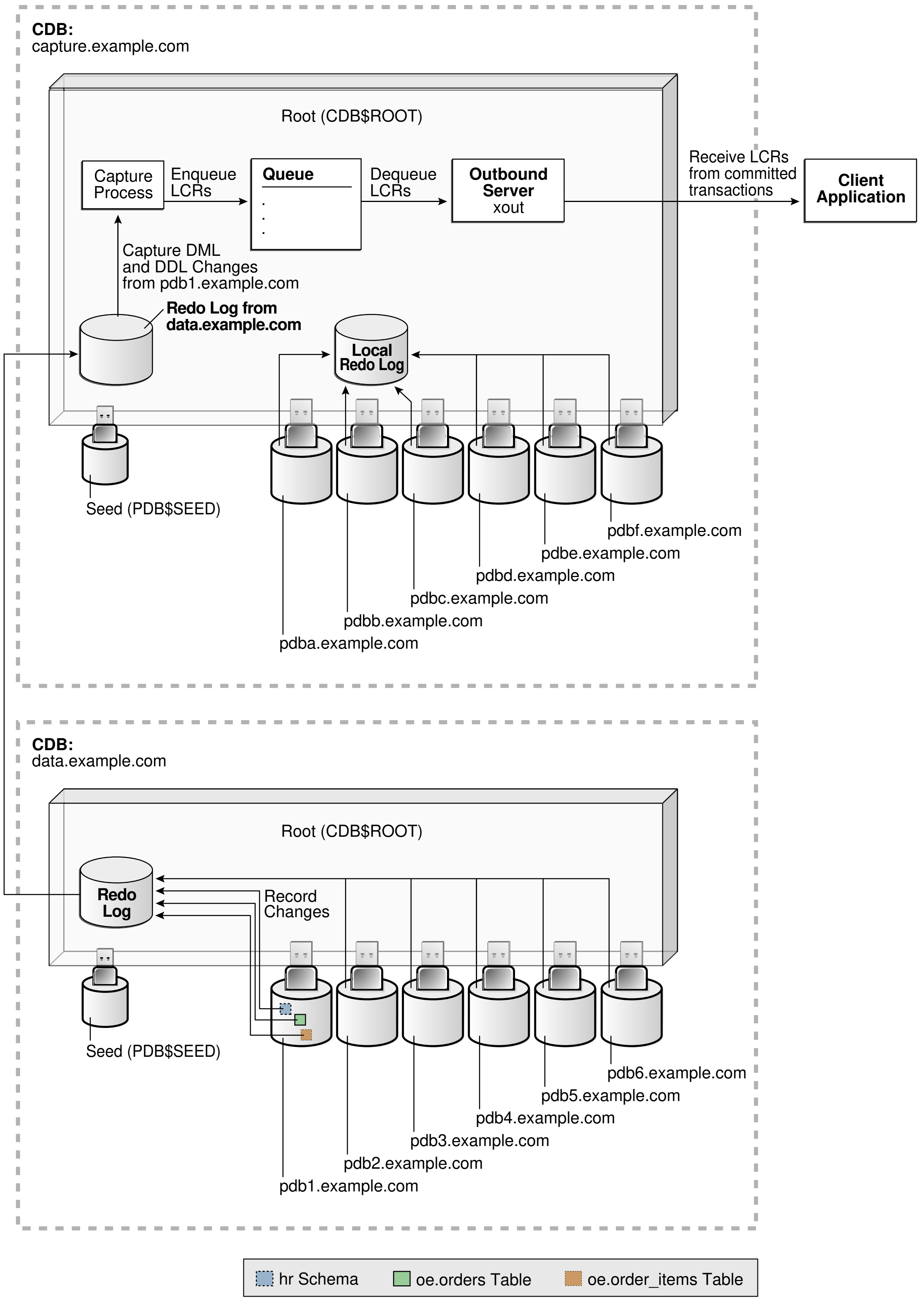
Assumptions

This section makes the following assumptions:

- The name of the outbound server is `xout`.
- The queue used by the outbound server is `c##xstrmadmin.xstream_queue`.
- The source database is the PDB `pdb1.example.com` in the CDB `data.example.com`.
- The capture process runs in the CDB `capture.example.com`.
- The outbound server runs in the CDB `capture.example.com`.
- DML and DDL changes made to the `oe.orders` and `oe.order_items` tables from the PDB `pdb1.example.com` are sent to the outbound server.
- DML and DDL changes made to the `hr` schema from the PDB `pdb1.example.com` are sent to the outbound server.

The following figure gives an overview of this XStream Out configuration.

Figure 4-7   Sample XStream Out Configuration Using Multiple CDBs and Downstream Capture



To configure XStream Out with downstream capture in CDBs:

1. In SQL*Plus, connect to the root of the downstream capture CDB as the XStream administrator.

   In this example. the downstream capture CDB is `capture.example.com`.

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

   For example, run the following procedure:

   ```
   BEGIN
     DBMS_XSTREAM_ADM.SET_UP_QUEUE(
       queue_table => 'c##xstrmadmin.xstream_queue_table',
       queue_name  => 'c##xstrmadmin.xstream_queue');
   END;
   /
   ```

3. Optionally, create the database link from the root in the downstream capture CDB to the root in the source CDB.

   In this example, create a database link from the root in `capture.example.com` to the root in `data.example.com`. For example, if the user `c##xstrmadmin` is the XStream administrator on both databases, then create the following database link:

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

4. Ensure that all containers in the source CDB are in open read/write mode.

5. If you did not create the database link in Step 3, then you must complete additional steps in the root of the source CDB.

   These steps are not required if you created the database link in Step 3.

   Run the `BUILD` procedure and ensure that required supplemental logging is specified for the database objects in the source CDB:

   a. Connect to the root in the source CDB as the XStream administrator.

   b. Run the `DBMS_CAPTURE_ADM.BUILD` procedure. For example:

      ```
      SET SERVEROUTPUT ON
      DECLARE
        scn  NUMBER;
      BEGIN
        DBMS_CAPTURE_ADM.BUILD(
          first_scn => scn);
        DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
      END;
      /
      First SCN Value = 409391
      ```

      This procedure displays the valid first SCN value for the capture process that will be created in the root in the `capture.example.com` CDB. Make a note of the SCN value returned because you will use it when you create the capture process in Step 6.

   c. Ensure that required supplemental logging is specified for the database objects in the source CDB.

For this example, ensure that supplemental logging is configured for the `hr` schema, the `oe.orders` table, and the `oe.order_items` table in the `pdb1.example.com` PDB.

6. While connected to the root in the downstream capture CDB, create the capture process.

   For example, run the following procedure to create the capture process while connected as the XStream administrator to `capture.example.com`:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'c##xstrmadmin.xstream_queue',
    capture_name       => 'real_time_capture',
    rule_set_name      => NULL,
    start_scn          => NULL,
    source_database    => NULL,
    use_database_link  => TRUE,
    first_scn          => NULL,
    logfile_assignment => 'implicit',
    source_root_name   => 'data.example.com',
    capture_class      => 'xstream');
END;
/
```

   If you did not create a database link in Step 3, then specify the SCN value returned by the `DBMS_CAPTURE_ADM.BUILD` procedure for the `first_scn` parameter.

   Do not start the capture process.

7. After the capture process is created, optionally change the open mode of one or more PDBs if necessary.

8. 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            =>  'c##xstrmadmin.xstream_queue',
    source_database       =>  'pdb1.example.com',
    table_names           =>  tables,
    schema_names          =>  schemas,
    source_root_name      => 'data.example.com',
    source_container_name => 'pdb1');
END;
/
```

   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 PDB `pdb1.example.com` in the CDB `data.example.com`. The outbound server dequeues LCRs from the queue `c##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.

> **Note:**
>
> The `server_name` value cannot exceed 30 bytes.

9. Create and run the client application that will connect to the outbound server and receive the LCRs.

   When you run the client application, the outbound server is started automatically at the downstream capture CDB.

Related Topics

- Prerequisites for Configuring XStream Out
  Preparing for an XStream Out outbound server is similar to preparing for an Oracle Replication environment.

5

Managing XStream Out

You can manage XStream Out components and their rules.

- About Managing XStream Out
  You can modify the database components that are part of an XStream Out configuration, such as outbound servers, capture processes, and rules.
- Managing an Outbound Server
  You can manage an outbound server by starting it, stopping it, setting an apply parameter for it, and changing its connect user.
- Managing the Capture Process for an Outbound Server
  You can manage the capture process for an outbound server. The capture process captures database changes and sends them to an outbound server.
- Managing Rules for an XStream Out Configuration
  You can manage the 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.
- Managing Declarative Rule-Based Transformations
  Declarative rule-based transformations cover a set of common transformation scenarios for row LCRs.
- Dropping Components in an XStream Out Configuration
  To drop an outbound server, use the `DROP_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.
- Removing an XStream Out Configuration
  You run the `REMOVE_XSTREAM_CONFIGURATION` procedure in the `DBMS_XSTREAM_ADM` package to remove an XStream Out configuration in a multitenant container database (CDB) or non-CDB.

About Managing XStream Out

You can modify the database components that are part of an XStream Out configuration, such as outbound servers, capture processes, and rules.

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

- `DBMS_XSTREAM_ADM`

  The `DBMS_XSTREAM_ADM` package is the main package for managing XStream Out. This package includes subprograms that enable you to configure, modify, or drop outbound servers. This package also enables you modify the rules used by capture processes and outbound servers.

> **Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for detailed information about this package

- `XSTREAM_CAPTURE`

  The `XSTREAM_CAPTURE` role enables you to configure and modify XStream administrators.

  Starting with Oracle Database 23ai, you can manage XStream administrators with the `XSTREAM_CAPTURE` role. You can use the `REVOKE_ADMIN_PRIVILEGE` procedure to revoke privileges from a user that received XStream privileges prior to the Oracle Database 23ai upgrade.

  > **See Also:**
  >
  > – Configure an XStream Administrator on All Databases for information on creating an XStream administrator
  >
  > – *Oracle Database PL/SQL Packages and Types Reference* for detailed information about this package

- `DBMS_APPLY_ADM`

  The `DBMS_APPLY_ADM` package enables you modify outbound servers.

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

Managing an Outbound Server

You can manage an outbound server by starting it, stopping it, setting an apply parameter for it, and changing its connect user.

- Starting an Outbound Server
  A outbound server must be enabled for it to send logical change records (LCRs) to an XStream client application. You run the `START_OUTBOUND` procedure in the `DBMS_OUTBOUND_ADM` package to start an existing outbound server.

- Stopping an Outbound Server
  You run the `STOP_SERVER` procedure in the `DBMS_XSTREAM_ADM` package to stop an existing outbound server. You might stop an outbound server when you are troubleshooting a problem in an XStream configuration.
- Setting an Apply Parameter for an Outbound Server
  You set an apply parameter for an outbound server using the `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package. Apply parameters control the way an outbound server operates.
- Changing the Connect User for an Outbound Server
  A client application connects to an outbound server as the connect user. You can change the connect user for an outbound server using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

Starting an Outbound Server

A outbound server must be enabled for it to send logical change records (LCRs) to an XStream client application. You run the `START_OUTBOUND` procedure in the `DBMS_OUTBOUND_ADM` package to start an existing outbound server.

To start 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 `START_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package, and specify the outbound server for the `server_name` parameter.

The following example starts an outbound server named `xout`.

Example 5-1 Starting an Outbound Server Named xout

```
BEGIN
  DBMS_XSTREAM_ADM.START_OUTBOUND(
    server_name => 'xout');
END;
/
```

Note:

When an XStream client application attaches to an outbound server, it starts the outbound server and the outbound server's capture process automatically if either of these components are disabled.

See Also:

The Oracle Enterprise Manager Cloud Control online help for instructions about starting an apply process or an outbound server with Oracle Enterprise Manager Cloud Control

Stopping an Outbound Server

You run the `STOP_SERVER` procedure in the `DBMS_XSTREAM_ADM` package to stop an existing outbound server. You might stop an outbound server when you are troubleshooting a problem in an XStream configuration.

To stop 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 `STOP_SERVER` procedure in the `DBMS_XSTREAM_ADM` package, and specify the outbound server for the `server_name` parameter.

The following example stops an outbound server named `xout`.

Example 5-2 Stopping an Outbound Server Named xout

```
BEGIN
  DBMS_XSTREAM_ADM.STOP_OUTBOUND(
    server_name => 'xout');
END;
/
```

See Also:

The Oracle Enterprise Manager Cloud Control online help for instructions about stopping an apply process or an outbound server with Oracle Enterprise Manager Cloud Control

Setting an Apply Parameter for an Outbound Server

You set an apply parameter for an outbound server using the `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package. Apply parameters control the way an outbound server operates.

To set an outbound server parameter:

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 `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package.

Example 5-3 Setting an Outbound Server Parameter

The following example sets the `disable_on_error` parameter for an outbound server named `xout` to `N`.

```
BEGIN
  DBMS_XSTREAM_ADM.SET_PARAMETER(
    streams_name => 'xout',
    streams_type => 'apply',
    parameter    => 'disable_on_error',
    value         => 'N');
END;
/
```

Example 5-4 Setting an Outbound Server Parameter to Its Default Value

If the `value` parameter is set to `NULL` or is not specified, then the parameter is set to its default value. The following example sets the `MAX_SGA_SIZE` apply parameter to `NULL`:

```
BEGIN
  DBMS_XSTREAM_ADM.SET_PARAMETER(
    streams_name => 'xout',
    streams_type => 'apply',
    parameter    => 'max_sga_size',
    value        => NULL);
END;
/
```

Note:

- The value parameter is always entered as a `VARCHAR2` value, even if the parameter value is a number.
- If the `value` parameter is set to `NULL` or is not specified, then the parameter is set to its default value.

See Also:

- The Oracle Enterprise Manager Cloud Control online help for instructions about setting an apply parameter with Oracle Enterprise Manager Cloud Control
- *Oracle Database PL/SQL Packages and Types Reference* for information about apply parameters

Changing the Connect User for an Outbound Server

A client application connects to an outbound server as the connect user. You can change 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.

Note:

The default connect_user is the user that configured the outbound server. If you want to run the client application as a different user, follow the steps outlined below.

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 5-5   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:

- "Privileges Required by the Connect User for an Outbound Server"
- *Oracle Database PL/SQL Packages and Types Reference*

Managing the Capture Process for an Outbound Server

You can manage the capture process for an outbound server. The capture process captures database changes and sends them to an outbound server.

- 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.
- Starting a Capture Process
  A capture process must be enabled for it to capture database changes and send the changes to an XStream outbound server. You run the START_CAPTURE procedure in the DBMS_CAPTURE_ADM package to start an existing capture process.

- Stopping a Capture Process
  You run the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to stop an existing capture process. You might stop a capture process when you are troubleshooting a problem in an XStream configuration.
- Setting a Capture Process Parameter
  Capture process parameters control the way a capture process operates. You set a capture process parameter using the `SET_PARAMETER` procedure in the `DBMS_CAPTURE_ADM` package.
- Changing the Capture User of an Outbound Server's Capture Process
  A capture user is the user in whose security domain a capture process captures changes from the redo log.
- Changing the Start SCN or Start Time of an Outbound Server's Capture Process
  You can change 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.
- Setting the First SCN for a Capture Process
  You can set the first system change number (SCN) for an existing capture process. The first SCN is the SCN in the redo log from which a capture process can capture changes.

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.

Even when you cannot use the `DBMS_XSTREAM_ADM` package, you can always use the `DBMS_CAPTURE_ADM` package to manage the capture process.

The `DBMS_XSTREAM_ADM` package can manage an outbound server's capture process if either of the following conditions are met:

- 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.

If either of these conditions are met, then 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
- Set the start system change number (SCN) or start time
- Drop the capture process

The `DBMS_CAPTURE_ADM` package can manage a capture process in the following ways:

- Start and stop the capture process
- Alter the capture process, which includes changing the capture process's rule sets, capture user, first SCN, start SCN, and start time
- Set capture process parameters
- Drop the capture process

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 ALL_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. 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.

See Also:

- "Managing Rules for an XStream Out Configuration"
- "Changing the Capture User of an Outbound Server's Capture Process"
- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*

Starting a Capture Process

A capture process must be enabled for it to capture database changes and send the changes to an XStream outbound server. You run the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to start an existing capture process.

To start a capture process:

1. Connect to the capture process database as the XStream administrator.

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

2. Run the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package, and specify the capture process for the `capture_name` parameter.

The following example starts a capture process named `xstream_capture`.

Example 5-6 Starting a Capture Process Named xstream_capture

```
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'xstream_capture');
END;
/
```

Note:

When an XStream client application attaches to an outbound server, it starts the outbound server's capture process automatically if the capture process is disabled.

See Also:

The Oracle Enterprise Manager Cloud Control online help for instructions about starting a capture process with Oracle Enterprise Manager Cloud Control

Stopping a Capture Process

You run the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to stop an existing capture process. You might stop a capture process when you are troubleshooting a problem in an XStream configuration.

To stop a capture process:

1. Connect to the capture process database as the XStream administrator.

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

2. Run the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package, and specify the capture process for the `capture_name` parameter.

The following example starts a capture process named `xstream_capture`.

Example 5-7 Stopping a Capture Process Named xstream_capture

```
BEGIN
  DBMS_CAPTURE_ADM.STOP_CAPTURE(
    capture_name => 'xstream_capture');
END;
/
```

See Also:

The Oracle Enterprise Manager Cloud Control online help for instructions about stopping a capture process with Oracle Enterprise Manager Cloud Control

Setting a Capture Process Parameter

Capture process parameters control the way a capture process operates. You set a capture process parameter using the `SET_PARAMETER` procedure in the `DBMS_CAPTURE_ADM` package.

To set a capture process parameter:

1. Connect to the capture process database as the XStream administrator.

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

2. Run the `SET_PARAMETER` procedure in the `DBMS_CAPTURE_ADM` package.

The following example sets the `parallelism` parameter for a capture process named `xstream_capture` to `1` from the default value of `0`. The parallelism parameter controls the number of processes that concurrently mine the redo log for changes. It is a good idea to monitor the effect of increasing the parallelism for the capture process since additional processes are started.

Example 5-8 Setting a Capture Process Parameter

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'xstream_capture',
    parameter    => 'parallelism',
    value        => '1');
END;
/
```

Note:

- Setting the parallelism parameter automatically stops and restarts a capture process.
- The value parameter is always entered as a `VARCHAR2` value, even if the parameter value is a number.
- If the value parameter is set to `NULL` or is not specified, then the parameter is set to its default value.

See Also:

- The Oracle Enterprise Manager Cloud Control online help for instructions about setting a capture process parameter with Oracle Enterprise Manager Cloud Control
- *Oracle Database PL/SQL Packages and Types Reference* for information about capture process parameters

Changing the Capture User of an Outbound Server's Capture Process

A capture user is the user in whose security domain a capture process captures changes from the redo log.

You can change 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. Only a user granted `DBA` role can change the capture user for a capture process. 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:

Oracle Database PL/SQL Packages and Types Reference for information about the privileges required by a capture user

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

1. 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".

   If the capture process can be managed using the `DBMS_XSTREAM_ADM` package, then proceed to Step 2.

2. 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.

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:

Oracle Database PL/SQL Packages and Types Reference

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

You can change 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.

Note:

- The `start_scn` and `start_time` parameters in the `ALTER_OUTBOUND` procedure are mutually exclusive.
- You do not need to set the start SCN for a capture process after a normal restart of the database.

- Changing the Start SCN of an Outbound Server's Capture Process
  You can change the start SCN of the capture process for an outbound server.

- Changing the Start Time of an Outbound Server's Capture Process
  You can change the start time of the capture process for an outbound server.

Changing the Start SCN of an Outbound Server's Capture Process

You can change the start SCN of the capture process for an outbound server.

To change the start SCN for a capture process:

1. 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".

   If the capture process can be managed using the `DBMS_XSTREAM_ADM` package, then proceed to Step 2.

2. 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.

3. 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 ALL_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.

4. 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:

- *Oracle Database PL/SQL Packages and Types Reference*
- "SCN Values Related to a Capture Process"

Changing the Start Time of an Outbound Server's Capture Process

You can change the start time of the capture process for an outbound server.

To change the start time for a capture process:

1. 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".

   If the capture process can be managed using the `DBMS_XSTREAM_ADM` package, then proceed to Step 2.
2. 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.
3. 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 ALL_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.
4. 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:

- *Oracle Database PL/SQL Packages and Types Reference*
- "SCN Values Related to a Capture Process"

Setting the First SCN for a Capture Process

You can set the first system change number (SCN) for an existing capture process. The first SCN is the SCN in the redo log from which a capture process can capture changes.

The specified first SCN must meet the following requirements:

- It must be greater than the current first SCN for the capture process.
- It must be less than or equal to the current applied SCN for the capture process. However, this requirement does not apply if the current applied SCN for the capture process is zero.
- It must be less than or equal to the required checkpoint SCN for the capture process.

You can determine the current first SCN, applied SCN, and required checkpoint SCN for each capture process in a database using the following query:

```
SELECT CAPTURE_NAME, FIRST_SCN, APPLIED_SCN, REQUIRED_CHECKPOINT_SCN   FROM ALL_CAPTURE;
```

When you reset a first SCN for a capture process, information below the new first SCN setting is purged from the LogMiner data dictionary for the capture process automatically. Therefore, after the first SCN is reset for a capture process, the start SCN for the capture process cannot be set lower than the new first SCN. Also, redo log files that contain information before the new first SCN setting will never be needed by the capture process.

You set the first SCN for a capture process using the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

To set the first SCN for a capture process:

1. Connect to the capture process 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_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package, and specify the new first SCN in the `first_scn` parameter.

The following example sets the first SCN to `351232` for the `xstream_capture` capture process.

Example 5-13 Setting the First SCN for a Capture Process

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'xstream_capture',
    first_scn    => 351232);
END;
/
```

Note:

- If the specified first SCN is higher than the current start SCN for the capture process, then the start SCN is set automatically to the new value of the first SCN.
- If you must capture changes in the redo log from a point in time in the past, then you can create a capture process and specify a first SCN that corresponds to a previous data dictionary build in the redo log. The `BUILD` procedure in the `DBMS_CAPTURE_ADM` package performs a data dictionary build in the redo log.
- You can query the `DBA_LOGMNR_PURGED_LOG` data dictionary view to determine which redo log files will never be needed by any capture process.

See Also:

"SCN Values Related to a Capture Process"

Managing Rules for an XStream Out Configuration

You can manage the 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.

- Adding Rules to an XStream Out Configuration
  You can add schema rules, table rules, and subset rules to an XStream Out configuration.
- Removing Rules from an XStream Out Configuration
  You can remove rules from an XStream Out configuration.

Adding Rules to an XStream Out Configuration

You can add schema rules, table rules, and subset rules to an XStream Out configuration.

- Adding Schema Rules and Table Rules to an XStream Out Configuration
  You can add schema rules and table rules to an XStream Out configuration using the `ALTER_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package.
- Adding Subset Rules to an Outbound Server's Positive Rule Set
  You can add subset rules to an outbound server's positive rule set using the `ADD_SUBSET_OUTBOUND_RULES` procedure in the `DBMS_XSTREAM_ADM` package.
- Adding Rules With Custom Conditions to XStream Out Components
  Some of the procedures that create rules in the `DBMS_XSTREAM_ADM` package include an `and_condition` parameter. This parameter enables you to add conditions to system-created rules.

Adding Schema Rules and Table Rules to an XStream Out Configuration

You can add 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". 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 use the `ADD_*_RULES` procedures in the `DBMS_XSTREAM_ADM` package to add 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. Use the `ADD_*_PROPAGATION_RULES` procedures in the `DBMS_XSTREAM_ADM` package to add them.

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 5-14 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 5-15 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:

- "Rules and Rule Sets"
- *Oracle Database PL/SQL Packages and Types Reference*

Adding Subset Rules to an Outbound Server's Positive Rule Set

You can add 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 use the `ADD_*_RULES` procedures in the `DBMS_XSTREAM_ADM` package to add them.

Example 5-16 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 5-17 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:

- "Rules and Rule Sets"
- *Oracle Database PL/SQL Packages and Types Reference*
- "Declarative Rule-Based Transformations"

Adding Rules With Custom Conditions to XStream Out Components

Some of the procedures that create rules in the `DBMS_XSTREAM_ADM` package include an `and_condition` parameter. This parameter enables you to add conditions to system-created rules.

The condition specified by the `and_condition` parameter is appended to the system-created rule condition using an `AND` clause in the following way:

```
(system_condition) AND (and_condition)
```

The variable in the specified condition must be `:lcr`.

To add a rule with a custom condition to an XStream Out component:

1. Connect to the database running the XStream Out component as the XStream administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.
2. Run an `ADD_*_RULES` procedure and specify the custom condition in the `and_condition` parameter.

   See "System-Created Rules and XStream" for information about these procedures.

If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then ensure that this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify `TRUE` for the `include_dml` parameter and `FALSE` for the `include_ddl` parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify `FALSE` for the `include_dml` parameter and `TRUE` for the `include_ddl` parameter.

For example, the `GET_OBJECT_TYPE` member function only applies to DDL LCRs. Therefore, if you use this member function in an `and_condition`, then specify `FALSE` for the `include_dml` parameter and `TRUE` for the `include_ddl` parameter.

Example 5-18 Adding a Table Rule With a Custom Condition

This example specifies that the table rules generated by the `ADD_TABLE_RULES` procedure evaluate to `TRUE` only if the table is `hr.departments`, the source database is `dbs1.example.com`, and the tag value is the hexadecimal equivalent of `'02'`.

```
BEGIN
  DBMS_XSTREAM_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'xout_capture',
    queue_name          =>  'xstream_queue',
    include_dml         =>  TRUE,
    include_ddl         =>  TRUE,
    include_tagged_lcr  =>  TRUE,
    source_database     =>  'dbs1.example.com',
    inclusion_rule      =>  TRUE,
    and_condition       =>  ':lcr.get_tag() = HEXTORAW(''02'')');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about LCR member subprograms

Removing Rules from an XStream Out Configuration

You can remove rules from an XStream Out configuration.

- Removing Schema Rules and Table Rules From an XStream Out Configuration
  You can remove 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.
- Removing Subset Rules from an Outbound Server's Positive Rule Set
  You can remove subset rules from an outbound server's positive rule set using the `REMOVE_SUBSET_OUTBOUND_RULES` procedure in the `DBMS_XSTREAM_ADM` package.
- Removing Rules Using the REMOVE_RULE Procedure
  You can remove a single rule from an XStream Out component's rule set or all rules from the rule set using the `REMOVE_RULE` procedure in the `DBMS_XSTREAM_ADM` package.

Removing Schema Rules and Table Rules From an XStream Out Configuration

You can remove 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". 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 "Removing Rules Using the REMOVE_RULE Procedure" 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 "Removing Rules Using the REMOVE_RULE Procedure" 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 5-19 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 5-20 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:

- *Oracle Database PL/SQL Packages and Types Reference*
- "Declarative Rule-Based Transformations"

Removing Subset Rules from an Outbound Server's Positive Rule Set

You can remove 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 ALL_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
   ```

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 "Removing Rules Using the REMOVE_RULE Procedure" for information about removing rules.

Example 5-21 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;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference

Removing Rules Using the REMOVE_RULE Procedure

You can remove a single rule from an XStream Out component's rule set or all rules from the rule set using the `REMOVE_RULE` procedure in the `DBMS_XSTREAM_ADM` package.

The XStream Out component can be a capture process, propagation, or outbound server.

The `REMOVE_RULE` procedure only can remove rules for both DML and DDL changes, and it can remove rules from either the component's positive rule set or negative rule set.

To remove a single rule or all rules from an outbound server's rule set:

1. Connect to the database running the XStream Out component as the XStream administrator.

   See *Oracle Database Administrator’s Guide* for information about connecting to a database in SQL*Plus.
2. Determine the rule name and XStream component name.

   See "Monitoring XStream Rules" for a query that displays this information.
3. Run the `REMOVE_RULE` procedure.

The `inclusion_rule` parameter is set to `TRUE` to indicate the positive rule set.

The `rule_name` parameter is set to `NULL` to specify that all of the rules are removed from the rule set, and the `inclusion_rule` parameter is set to `FALSE` to indicate the negative rule set.

Example 5-22 Removing a Rule From an Outbound Server's Rule Set

This example removes a rule named `orders12` from positive rule set of the `xout` outbound server.

```
BEGIN
  DBMS_XSTREAM_ADM.REMOVE_RULE(
   rule_name      => 'orders12',
   streams_type   => 'APPLY',
   streams_name   => 'xout',
   inclusion_rule => TRUE);
/
```

Example 5-23 Removing All of the Rules From an Outbound Server's Rule Set

This example removes all of the rules from the negative rule set of the `xout` outbound server.

```
BEGIN
  DBMS_XSTREAM_ADM.REMOVE_RULE(
   rule_name      => NULL,
   streams_type   => 'APPLY',
   streams_name   => 'xout',
   inclusion_rule => FALSE);
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference

Managing Declarative Rule-Based Transformations

Declarative rule-based transformations cover a set of common transformation scenarios for row LCRs.

You can use the following procedures in the `DBMS_XSTREAM_ADM` package to manage declarative rule-based transformations: `ADD_COLUMN`, `DELETE_COLUMN`, `KEEP_COLUMNS`, `RENAME_COLUMN`, `RENAME_SCHEMA`, and `RENAME_TABLE`.

- Adding Declarative Rule-Based Transformations
  Examples illustrate adding declarative rule-based transformations to DML rules.
- Overwriting Existing Declarative Rule-Based Transformations
  You can overwrite existing declarative rule-based transformations using the `DBMS_XSTREAM_ADM` package.
- Removing Declarative Rule-Based Transformations
  To remove a declarative rule-based transformation from a rule, use the same procedure used to add the transformation, but specify `REMOVE` for the `operation` parameter.

See Also:

"Declarative Rule-Based Transformations"

Adding Declarative Rule-Based Transformations

Examples illustrate adding declarative rule-based transformations to DML rules.

Note:

Declarative rule-based transformations can be specified for DML rules only. They cannot be specified for DDL rules.

- Adding a Declarative Rule-Based Transformation That Renames a Table
  Use the `RENAME_TABLE` procedure in the `DBMS_XSTREAM_ADM` package to add a declarative rule-based transformation that renames a table in a row LCR.
- Adding a Declarative Rule-Based Transformation That Adds a Column
  Use the `ADD_COLUMN` procedure in the `DBMS_XSTREAM_ADM` package to add a declarative rule-based transformation that adds a column to a row in a row LCR.

Adding a Declarative Rule-Based Transformation That Renames a Table

Use the `RENAME_TABLE` procedure in the `DBMS_XSTREAM_ADM` package to add a declarative rule-based transformation that renames a table in a row LCR.

The example in this section adds a declarative rule-based transformation to the `jobs12` rule in the `xstrmadmin` schema.

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 procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.RENAME_TABLE(
    rule_name       => 'xstrmadmin.jobs12',
    from_table_name => 'hr.jobs',
    to_table_name   => 'hr.assignments',
    step_number     => 0,
    operation       => 'ADD');
END;
/
```

The declarative rule-based transformation added by this procedure renames the table `hr.jobs` to `hr.assignments` in a row LCR when the rule `jobs12` evaluates to `TRUE` for the row LCR. If more than one declarative rule-based transformation is specified for the `jobs12` rule, then this transformation follows default transformation ordering because the `step_number` parameter is set to `0` (zero). In addition, the `operation` parameter is set to `ADD` to indicate that the transformation is being added to the rule, not removed from it.

The `RENAME_TABLE` procedure can also add a transformation that renames the schema in addition to the table. For example, in the previous example, to specify that the schema should be renamed to `oe`, specify `oe.assignments` for the `to_table_name` parameter.

Adding a Declarative Rule-Based Transformation That Adds a Column

Use the `ADD_COLUMN` procedure in the `DBMS_XSTREAM_ADM` package to add a declarative rule-based transformation that adds a column to a row in a row LCR.

The example in this section adds a declarative rule-based transformation to the `employees35` rule in the `xstrmadmin` schema.

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 procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.ADD_COLUMN(
    rule_name    => 'xstrmadmin.employees35',
    table_name   => 'hr.employees',
    column_name  => 'birth_date',
    column_value => ANYDATA.ConvertDate(NULL),
    value_type   => 'NEW',
    step_number  => 0,
    operation    => 'ADD');
```

```
END;
/
```

The declarative rule-based transformation added by this procedure adds a `birth_date` column of data type `DATE` to an `hr.employees` table row in a row LCR when the rule `employees35` evaluates to `TRUE` for the row LCR.

Notice that the `ANYDATA.ConvertDate` function specifies the column type and the column value. In this example, the added column value is `NULL`, but a valid date can also be specified. Use the appropriate `ANYDATA` function for the column being added. For example, if the data type of the column being added is `NUMBER`, then use the `ANYDATA.ConvertNumber` function.

The `value_type` parameter is set to `NEW` to indicate that the column is added to the new values in a row LCR. You can also specify `OLD` to add the column to the old values.

If more than one declarative rule-based transformation is specified for the `employees35` rule, then the transformation follows default transformation ordering because the `step_number` parameter is set to `0` (zero). In addition, the `operation` parameter is set to `ADD` to indicate that the transformation is being added, not removed.

Note:

The `ADD_COLUMN` procedure is overloaded. A `column_function` parameter can specify that the current system date or time stamp is the value for the added column. The `column_value` and `column_function` parameters are mutually exclusive.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about `AnyData` type functions

Overwriting Existing Declarative Rule-Based Transformations

You can overwrite existing declarative rule-based transformations using the `DBMS_XSTREAM_ADM` package.

When the `operation` parameter is set to `ADD` in a procedure that adds a declarative rule-based transformation, an existing declarative rule-based transformation is overwritten if the parameters in the following list match the existing transformation parameters:

- `ADD_COLUMN` procedure: `rule_name`, `table_name`, `column_name`, and `step_number` parameters
- `DELETE_COLUMN` procedure: `rule_name`, `table_name`, `column_name`, and `step_number` parameters
- `KEEP_COLUMNS` procedure: `rule_name`, `table_name`, `column_list`, and `step_number` parameters, or `rule_name`, `table_name`, `column_table`, and `step_number` parameters (The `column_list` and `column_table` parameters are mutually exclusive.)
- `RENAME_COLUMN` procedure: `rule_name`, `table_name`, `from_column_name`, and `step_number` parameters

- `RENAME_SCHEMA` procedure: `rule_name`, `from_schema_name`, and `step_number` parameters
- `RENAME_TABLE` procedure: `rule_name`, `from_table_name`, and `step_number` parameters

To overwrite an existing rule-based transformation:

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 appropriate procedure in the `DBMS_XSTREAM_ADM` package, and specify the appropriate parameters.

Example 5-24 Overwriting a RENAME_COLUMN Declarative Rule-Based Transformation

Suppose an existing declarative rule-based transformation was creating by running the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.RENAME_COLUMN(
    rule_name         => 'departments33',
    table_name        => 'hr.departments',
    from_column_name  => 'manager_id',
    to_column_name    => 'lead_id',
    value_type        => 'NEW',
    step_number       => 0,
    operation         => 'ADD');
END;
/
```

Running the following procedure overwrites this existing declarative rule-based transformation:

```
BEGIN
  DBMS_XSTREAM_ADM.RENAME_COLUMN(
    rule_name         => 'departments33',
    table_name        => 'hr.departments',
    from_column_name  => 'manager_id',
    to_column_name    => 'lead_id',
    value_type        => '*',
    step_number       => 0,
    operation         => 'ADD');
END;
/
```

In this case, the `value_type` parameter in the declarative rule-based transformation was changed from `NEW` to `*`. That is, in the original transformation, only new values were renamed in row LCRs, but, in the new transformation, both old and new values are renamed in row LCRs.

Removing Declarative Rule-Based Transformations

To remove a declarative rule-based transformation from a rule, use the same procedure used to add the transformation, but specify `REMOVE` for the `operation` parameter.

To remove a declarative rule-based transformation:

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 appropriate procedure in the DBMS_XSTREAM_ADM package and specify REMOVE for the operation parameter.

When the operation parameter is set to REMOVE in any of the declarative transformation procedures listed in "Managing Declarative Rule-Based Transformations", the other parameters in the procedure are optional, excluding the rule_name parameter. If these optional parameters are set to NULL, then they become wildcards.

The RENAME_TABLE procedure in the previous example behaves in the following way when one or more of the optional parameters are set to NULL:

Table 5-1 Behavior of Optional Parameters in the RENAME_TABLE Procedure

from_table_name Parameter	to_table_name Parameter	step_number Parameter	Result
NULL	NULL	NULL	Remove all rename table transformations for the specified rule
non-NULL	NULL	NULL	Remove all rename table transformations with the specified from_table_name for the specified rule
NULL	non-NULL	NULL	Remove all rename table transformations with the specified to_table_name for the specified rule
NULL	NULL	non-NULL	Remove all rename table transformations with the specified step_number for the specified rule
non-NULL	non-NULL	NULL	Remove all rename table transformations with the specified from_table_name and to_table_name for the specified rule
NULL	non-NULL	non-NULL	Remove all rename table transformations with the specified to_table_name and step_number for the specified rule
non-NULL	NULL	non-NULL	Remove all rename table transformations with the specified from_table_name and step_number for the specified rule

The other declarative transformation procedures work in a similar way when optional parameters are set to NULL and the operation parameter is set to REMOVE.

Example 5-25 Removing a RENAME_TABLE Declarative Rule-Based Transformation

To remove the transformation added in "Adding a Declarative Rule-Based Transformation That Renames a Table", run the following procedure:

```
BEGIN
  DBMS_XSTREAM_ADM.RENAME_TABLE(
    rule_name       => 'strmadmin.jobs12',
    from_table_name => 'hr.jobs',
    to_table_name   => 'hr.assignments',
    step_number     => 0,
    operation       => 'REMOVE');
END;
/
```

Dropping Components in an XStream Out Configuration

To drop an outbound server, use 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".

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 5-26 Dropping an Outbound Server

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

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

See Also:

Oracle Database PL/SQL Packages and Types Reference for information about the `DROP_OUTBOUND` procedure

Removing an XStream Out Configuration

You run the `REMOVE_XSTREAM_CONFIGURATION` procedure in the `DBMS_XSTREAM_ADM` package to remove an XStream Out configuration in a multitenant container database (CDB) or non-CDB.

> **Note:**
>
> Run this procedure only if you are sure you want to remove the entire XStream Out configuration at a database. This procedure also removes all XStream In components, Oracle GoldenGate components, and Oracle Replication components from the database.

To remove the XStream Out configuration:

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

   In a CDB, connect to the CDB root.

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

2. Run the `REMOVE_XSTREAM_CONFIGURATION` procedure.

   In a non-CDB, run the following procedure:

   ```
   EXEC DBMS_XSTREAM_ADM.REMOVE_XSTREAM_CONFIGURATION();
   ```

   In a CDB, ensure that all containers are open in read/write mode and run the following procedure:

   ```
   EXEC DBMS_XSTREAM_ADM.REMOVE_XSTREAM_CONFIGURATION(container => 'ALL');
   ```

   Setting the `container` parameter to `ALL` removes the XStream configuration from all containers in the CDB.

3. Drop the XStream administrator at the database, if possible.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the actions performed by the `REMOVE_XSTREAM_CONFIGURATION` procedure

6 Monitoring XStream Out

You can monitor an XStream Out configuration.

Note:

Whenever possible, this chapter uses `ALL_` static data dictionary views for query examples. In some cases, information in the `ALL_` views is more limited than the information in the `DBA_` views.

- About Monitoring XStream Out
  You can query data dictionary views related to XStream for information about XStream components and statistics related to XStream.
- Monitoring Session Information About XStream Out Components
  An example illustrates monitoring session information about XStream Out components.
- Monitoring the History of Events for XStream Out Components
  An example illustrates monitoring the history of events for XStream components by querying the `DBA_REPLICATION_PROCESS_EVENTS` view.
- Monitoring an Outbound Server
  Sample queries illustrate how to monitor an outbound server.
- Monitoring the Capture Process for an Outbound Server
  Sample queries illustrate how to monitor the capture process for an outbound server.
- Monitoring XStream Rules
  A sample query illustrates how to monitor XStream rules.
- Monitoring Declarative Rule-Based Transformations
  A sample query illustrates how to monitor declarative rule-based transformations.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- "Configuring XStream Out"
- "Troubleshooting XStream Out"

About Monitoring XStream Out

You can query data dictionary views related to XStream for 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 Cloud Control. For example, you can view information about capture processes, outbound servers, inbound servers, and rules in Oracle Enterprise Manager Cloud Control.

In SQL*Plus, trusted XStream administrators can query the `ALL_` and `DBA_` views. Untrusted XStream administrators can query the `ALL_` views only.

This chapter also describes using the Oracle Replication Performance Advisor to monitor an XStream configuration. The Oracle Replication Performance Advisor consists a collection of data dictionary views. The Oracle Replication Performance Advisor enables you to monitor the topology and performance of an XStream environment.

Monitoring Session Information About XStream Out Components

An example illustrates monitoring session information about XStream Out components.

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 program name

This query is especially useful for determining the session information for specific XStream components when there are multiple XStream Out 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 ID' FORMAT A17
COLUMN PROCESS_NAME HEADING 'XStream|Program|Name' 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  Program
XStream Component              Session ID    Number Process ID        Name
------------------------------ ---------- --------- ----------------- -------
```

```
XOUT - Apply Coordinator                21         9 27222             AP01
CAP$_XOUT_18 - Capture                  28        33 27248             CP01
XOUT - Apply Server                     97        43 27226             AS00
XOUT - Apply Reader                    105         5 27224             AS01
XOUT - Apply Server                    112        27 27342             TNS
XOUT - Propagation Send/Rcv            117         5 27250             CS00
```

The row that shows `TNS` for the XStream program name contains information about the session for the XStream client application that is attached to the outbound server.

See Also:

Oracle Database Reference for more information about the `V$SESSION` view

Monitoring the History of Events for XStream Out Components

An example illustrates monitoring the history of events for XStream components by querying the `DBA_REPLICATION_PROCESS_EVENTS` view.

For example, the view can display when a component was created or started. It can also display when a component parameter was changed. If the component encountered an error, the view can display information about the error.

The query in this topic displays the following information about XStream Out component events:

- The XStream component name
- The component type
- The event name
- The description of the event
- The event time

To display this information for each XStream Out 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 STREAMS_NAME FORMAT A12
COLUMN PROCESS_TYPE FORMAT A17
COLUMN EVENT_NAME FORMAT A10
COLUMN DESCRIPTION FORMAT A20
COLUMN EVENT_TIME FORMAT A15

SELECT STREAMS_NAME,
       PROCESS_TYPE,
       EVENT_NAME,
       DESCRIPTION,
       EVENT_TIME
  FROM DBA_REPLICATION_PROCESS_EVENTS;
```

Your output for an XStream Out configuration looks similar to the following:

```
STREAMS_NAME PROCESS_TYPE      EVENT_NAME DESCRIPTION          EVENT_TIME
------------ ----------------- ---------- -------------------- ---------------
CAP$_XOUT_7  CAPTURE           CREATE     SUCCESS              10-NOV-15 12.30
                                                               .13.845080 PM
XOUT         APPLY COORDINATOR CREATE     SUCCESS              10-NOV-15 12.30
                                                               .16.841110 PM
"CAP$_XOUT_7 CAPTURE           ALTER      RULE_SET_NAME => "SY 10-NOV-15 12.30
"                                         S"."RULESET$_12"     .17.373285 PM
"XOUT"       APPLY COORDINATOR ALTER      RULE_SET_NAME => "SY 10-NOV-15 12.30
                                          S"."RULESET$_19"     .18.817718 PM
"CAP$_XOUT_7 CAPTURE           PARAMETER  Change parameter 'XO 10-NOV-15 12.30
"                              CHANGE     UT_CLIENT_EXISTS' to .19.100361 PM
                                           value 'Y'
CAP$_XOUT_7  CAPTURE           START      SUCCESS              10-NOV-15 12.30
                                                               .19.434029 PM
XOUT         APPLY COORDINATOR START      SUCCESS              10-NOV-15 12.30
                                                               .19.543379 PM
XOUT         APPLY READER      START      SUCCESS              10-NOV-15 12.30
                                                               .20.584332 PM
XOUT         APPLY SERVER      START      SUCCESS              10-NOV-15 12.30
                                                               .20.593923 PM
CAP$_XOUT_7  CAPTURE SERVER    START      SUCCESS              10-NOV-15 12.30
                                                               .20.926374 PM
```

Related Topics

- *Oracle Database Reference*

Monitoring an Outbound Server

Sample queries illustrate how to monitor an outbound server.

With XStream Out, an Oracle 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.

- Displaying General Information About an Outbound Server
  A sample query illustrates how to display general information about an outbound server.
- Displaying Status and Error Information for an Outbound Server
  A sample query illustrates how to display status and error information for an outbound server.
- Displaying Information About an Outbound Server's Current Transaction
  A sample query illustrates how to display information about an outbound server’s current transaction.
- Displaying Statistics for an Outbound Server
  An example illustrates how to display statistics for an outbound server.
- Displaying the Processed Low Position for an Outbound Server
  A sample query illustrates how to display the processed low position for an outbound server.
- Determining the Process Information for an Outbound Server
  A sample query illustrates how to determine the process information for an outbound server.
- Displaying the Apply Parameter Settings for an Outbound Server
  A sample query illustrates how to display the apply parameter settings for an outbound server.

Displaying General Information About an Outbound Server

A sample query illustrates how to display 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 `ALL_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 `ALL_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 A12
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 ALL_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$_XOUT_18 XOUT.EXAMPL XSTRMADMIN Q$_XOUT_19
                                              E.COM
```

See Also:

Oracle Database Reference

Displaying Status and Error Information for an Outbound Server

A sample query illustrates how to display status and error information for an outbound server.

The `DBA_APPLY` view shows `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 'Outbound Server|Name' FORMAT A15
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:

```
Outbound Server
Name            Status   Error Number Error Message
--------------- -------- ------------ ----------------------------------------
XOUT            ENABLED
```

This output shows that `XOUT` is an apply process that is functioning as an outbound server.

Note:

This example queries the `DBA_APPLY` view. This view enables trusted users to see information for all apply users in the database. Untrusted users must query the `ALL_APPLY` view, which limits information to the current user.

See Also:

Oracle Database Reference

Displaying Information About an Outbound Server's Current Transaction

A sample query illustrates how to display information about an outbound server's current transaction.

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       2.22.304            820023 0000000C82E4000 0000000C8337000        616
                                      000010000000100 000010000000100
                                      00000C82E400000 00000C833700000
                                      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:

Oracle Database Reference

Displaying Statistics for an Outbound Server

An example illustrates how to display statistics for an outbound server.

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 A9

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,
       TO_CHAR(LAST_SENT_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
          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          4028       256632               67                1    820023 10:11:00
                                                                          02/28/11
```

Note:

The `TOTAL_TRANSACTIONS_SENT` value is populated only if the `COMMITTED_DATA_ONLY` value is `YES` in `V$XSTREAM_OUTBOUND_SERVER`.

See Also:

Oracle Database Reference

Displaying the Processed Low Position for an Outbound Server

A sample query illustrates how to display 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 ALL_XSTREAM_OUTBOUND_PROGRESS;
   ```

Your output looks similar to the following:

```
Outbound                                Processed                      Processed
Server     Source                       Low LCR                        Low
Name       Database                     Position                       Time
---------- -------------------- ------------------------------ ---------
XOUT       XOUT.EXAMPLE.COM             0000000C84EA000000000000000000 10:18:37
                                        00000C84EA000000000000000001   02/28/11
```

See Also:

- *Oracle Database Reference*
- "The Processed Low Position and Restartability for XStream Out"

Determining the Process Information for an Outbound Server

A sample query illustrates how to determine the process information for an outbound server.

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                          18            19 15906
```

Note:

The `V$XSTREAM_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:

Oracle Database Reference

Displaying the Apply Parameter Settings for an Outbound Server

A sample query illustrates how to display the apply parameter settings for an outbound server.

Apply parameters determine how an outbound server operates.

To display the apply parameter settings 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 APPLY_NAME HEADING 'Outbound Server|Name' FORMAT A15
COLUMN PARAMETER HEADING 'Parameter' FORMAT A30
COLUMN VALUE HEADING 'Value' FORMAT A22
COLUMN SET_BY_USER HEADING 'Set by|User?' FORMAT A10

SELECT APPLY_NAME,
       PARAMETER,
       VALUE,
       SET_BY_USER
  FROM ALL_APPLY_PARAMETERS a, ALL_XSTREAM_OUTBOUND o
  WHERE a.APPLY_NAME=o.SERVER_NAME
  ORDER BY a.PARAMETER;
```

Your output looks similar to the following:

```
Outbound Server                                                       Set by
Name            Parameter                      Value                  User?
--------------- ------------------------------ ---------------------- ----------
XOUT            ALLOW_DUPLICATE_ROWS           N                      NO
XOUT            APPLY_SEQUENCE_NEXTVAL         Y                      NO
XOUT            COMMIT_SERIALIZATION           DEPENDENT_TRANSACTIONS NO
XOUT            COMPARE_KEY_ONLY               N                      NO
XOUT            COMPUTE_LCR_DEP_ON_ARRIVAL     N                      NO
XOUT            DISABLE_ON_ERROR               Y                      NO
XOUT            DISABLE_ON_LIMIT               N                      NO
XOUT            EAGER_SIZE                     9500                   NO
XOUT            ENABLE_XSTREAM_TABLE_STATS     Y                      NO
XOUT            EXCLUDETAG                                            NO
XOUT            EXCLUDETRANS                                          NO
XOUT            EXCLUDEUSER                                           NO
XOUT            EXCLUDEUSERID                                         NO
XOUT            GETAPPLOPS                     Y                      NO
XOUT            GETREPLICATES                  N                      NO
XOUT            GROUPTRANSOPS                  10000                  NO
XOUT            HANDLECOLLISIONS               N                      NO
XOUT            IGNORE_TRANSACTION                                    NO
XOUT            MAXIMUM_SCN                    INFINITE               NO
XOUT            MAX_PARALLELISM                1                      NO
XOUT            MAX_SGA_SIZE                   INFINITE               NO
XOUT            OPTIMIZE_PROGRESS_TABLE        Y                      NO
XOUT            OPTIMIZE_SELF_UPDATES          Y                      NO
XOUT            PARALLELISM                    1                      NO
XOUT            PRESERVE_ENCRYPTION            Y                      NO
XOUT            RTRIM_ON_IMPLICIT_CONVERSION   Y                      NO
XOUT            STARTUP_SECONDS                0                      NO
XOUT            SUPPRESSTRIGGERS               Y                      NO
XOUT            TIME_LIMIT                     INFINITE               NO
XOUT            TRACE_LEVEL                    0                      NO
XOUT            TRANSACTION_LIMIT              INFINITE               NO
XOUT            TXN_AGE_SPILL_THRESHOLD        900                    NO
XOUT            TXN_LCR_SPILL_THRESHOLD        10000                  NO
XOUT            WRITE_ALERT_LOG                Y                      NO
```

Note:

The apply parameter `OPTIMIZE_PROGRESS_TABLE` for Oracle GoldenGate Integrated Replicat, XStream In, and Logical Standby, is desupported in Oracle Database 19c. Before you upgrade to Oracle Database 19, you must turn off this parameter. If `OPTIMIZE_PROGRESS_TABLE` is set to ON, then stop apply gracefully, turn off the parameter, and restart apply. For GoldenGate Apply and XStream, this parameter is set to `OFF` by default.

Outbound servers ignore some apply parameter settings.

Note:

If the `Set by User?` column is `NO` for a parameter, then the parameter is set to its default value. If the `Set by User?` column is `YES` for a parameter, then the parameter was set by a user and might or might not be set to its default value.

See Also:

- "Setting an Apply Parameter for an Outbound Server"
- *Oracle Database PL/SQL Packages and Types Reference* for information about apply parameters

Monitoring the Capture Process for an Outbound Server

Sample queries illustrate how to monitor the capture process for an outbound server.

- Displaying Change Capture Information About Each Capture Process
  A sample query illustrates how to display change capture information about each capture process.
- Displaying the Registered Redo Log Files for Each Capture Process
  A sample query illustrates how to display information about the archived redo log files that are registered for each capture process in a database.
- Displaying Redo Log Files That Are Required by Each Capture Process
  A sample query illustrates how to display redo log files that are required by each capture process.
- Displaying SCN Values for Each Redo Log File Used by Each Capture Process
  A sample query illustrates how to display information about the SCN values for archived redo log files that are registered for each capture process in a database.
- Listing the Parameter Settings for Each Capture Process
  A sample query illustrates how to list the parameter settings for each capture process.
- Determining the Applied SCN for Each Capture Process
  A sample query illustrates how to determine the applied SCN for each capture process.

- Displaying the Redo Log Scanning Latency for Each Capture Process
  A sample query illustrates how to display the redo log scanning latency for each capture process.
- Displaying the Extra Attributes Captured by a Capture Process
  A sample query illustrates how to display the extra attributes captured by a capture process.

Displaying Change Capture Information About Each Capture Process

A sample query illustrates how to display change capture information about each capture process.

The query in this section displays the following information about each capture process in a database:

- The name of the capture process.
- The current state of the capture process

  See "Capture Process States".
- The total number of redo entries passed by LogMiner to the capture process for detailed rule evaluation. A capture process converts a redo entry into an LCR and performs detailed rule evaluation on the LCR when capture process prefiltering cannot discard the change.
- The total number LCRs enqueued since the capture process was last started.

To display this change capture information about each capture process 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 CAPTURE_NAME HEADING 'Capture|Name' FORMAT A15
COLUMN STATE HEADING 'State' FORMAT A25
COLUMN TOTAL_MESSAGES_CAPTURED HEADING 'Redo|Entries|Evaluated|In Detail' FORMAT
99999999999999
COLUMN TOTAL_MESSAGES_ENQUEUED HEADING 'Total|LCRs|Enqueued' FORMAT 99999999999999

SELECT CAPTURE_NAME,
       STATE,
       TOTAL_MESSAGES_CAPTURED,
       TOTAL_MESSAGES_ENQUEUED
  FROM V$XSTREAM_CAPTURE;
```

Your output looks similar to the following:

```
                                                     Redo
                                                  Entries           Total
Capture                                         Evaluated            LCRs
Name            State                           In Detail        Enqueued
--------------- ------------------------- --------------- ---------------
CAP$_XOUT_1     WAITING FOR TRANSACTION            297666          261798
```

The number of redo entries scanned can be higher than the number of DML and DDL redo entries captured by a capture process. Only DML and DDL redo entries that satisfy the rule sets of a capture process are captured and sent to an outbound server. Also, the total LCRs enqueued includes LCRs that contain transaction control statements. These row LCRs contain directives such as `COMMIT` and `ROLLBACK`. Therefore, the total LCRs enqueued is a number higher than the number of row changes and DDL changes enqueued by a capture process.

See Also:

"Row LCRs" for more information about transaction control statements

Displaying the Registered Redo Log Files for Each Capture Process

A sample query illustrates how to display information about the archived redo log files that are registered for each capture process in a database.

The sample query displays information about these files for both local capture processes and downstream capture processes.

The query displays the following information for each registered archived redo log file:

- The name of a capture process that uses the file
- The source database of the file
- The sequence number of the file
- The name and location of the file at the local site
- Whether the file contains the beginning of a data dictionary build
- Whether the file contains the end of a data dictionary build

To display the registered redo log files for each capture process:

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 CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A10
COLUMN SEQUENCE# HEADING 'Sequence|Number' FORMAT 99999
COLUMN NAME HEADING 'Archived Redo Log|File Name' FORMAT A20
COLUMN DICTIONARY_BEGIN HEADING 'Dictionary|Build|Begin' FORMAT A10
COLUMN DICTIONARY_END HEADING 'Dictionary|Build|End' FORMAT A10

SELECT r.CONSUMER_NAME,
       r.SOURCE_DATABASE,
       r.SEQUENCE#,
       r.NAME,
       r.DICTIONARY_BEGIN,
       r.DICTIONARY_END
  FROM DBA_REGISTERED_ARCHIVED_LOG r, ALL_CAPTURE c
  WHERE r.CONSUMER_NAME = c.CAPTURE_NAME;
```

Your output looks similar to the following:

```
                                                         Dictionary Dictionary
Capture                                                  Build      Build
Process         Source     Sequence Archived Redo Log    Begin      End
Name            Database     Number File Name
--------------- ---------- -------- -------------------- ---------- ----------
```

```
CAP$_XOUT_1      DBS2.EXAMP       15 /orc/dbs/log/arch2_1 NO          NO
                 LE.COM             _15_478347508.arc
CAP$_XOUT_1      DBS2.EXAMP       16 /orc/dbs/log/arch2_1 NO          NO
                 LE.COM             _16_478347508.arc
CAP$_XOUT_2      DBS1.EXAMP       45 /remote_logs/arch1_1 YES         YES
                 LE.COM             _45_478347335.arc
CAP$_XOUT_2      DBS1.EXAMP       46 /remote_logs/arch1_1 NO          NO
                 LE.COM             _46_478347335.arc
CAP$_XOUT_2      DBS1.EXAMP       47 /remote_logs/arch1_1 NO          NO
                 LE.COM             _47_478347335.arc
```

Assume that this query was run at the `dbs2.example.com` database, and that `cap$_xout_1` is a local capture process, and `cap$_xout_2` is a downstream capture process. The source database for the `cap$_xout_2` downstream capture process is `dbs1.example.com`. This query shows that there are two registered archived redo log files for `cap$_xout_1` and three registered archived redo log files for `cap$_xout_2`. This query shows the name and location of each of these files in the local file system.

See Also:

- "Capture Process Overview"
- "Local Capture and Downstream Capture"
- "SCN Values Related to a Capture Process" for information about dictionary builds

Displaying Redo Log Files That Are Required by Each Capture Process

A sample query illustrates how to display redo log files that are required by each capture process.

A capture process needs the redo log file that includes the required checkpoint SCN, and all subsequent redo log files. You can query the `REQUIRED_CHECKPOINT_SCN` column in the `ALL_CAPTURE` data dictionary view to determine the required checkpoint SCN for a capture process. Redo log files before the redo log file that contains the required checkpoint SCN are no longer needed by the capture process. These redo log files can be stored offline if they are no longer needed for any other purpose. If you reset the start SCN for a capture process to a lower value in the future, then these redo log files might be needed.

The query displays the following information for each required archived redo log file:

- The name of a capture process that uses the file
- The source database of the file
- The sequence number of the file
- The name and location of the required redo log file at the local site

To display this information about each required archive redo log file in a database, run the following query:

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 CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A10
COLUMN SEQUENCE# HEADING 'Sequence|Number' FORMAT 99999
COLUMN NAME HEADING 'Required|Archived Redo Log|File Name' FORMAT A40

SELECT r.CONSUMER_NAME,
       r.SOURCE_DATABASE,
       r.SEQUENCE#,
       r.NAME
  FROM DBA_REGISTERED_ARCHIVED_LOG r, ALL_CAPTURE c
  WHERE r.CONSUMER_NAME =  c.CAPTURE_NAME AND
        r.NEXT_SCN      >= c.REQUIRED_CHECKPOINT_SCN;
```

Your output looks similar to the following:

```
Capture                                    Required
Process         Source     Sequence Archived Redo Log
Name            Database     Number File Name
--------------- ---------- -------- ----------------------------------------
CAP$_XOUT_1     DBS2.EXAMP       16 /orc/dbs/log/arch2_1_16_478347508.arc
                LE.COM
CAP$_XOUT_2     DBS1.EXAMP       47 /remote_logs/arch1_1_47_478347335.arc
                LE.COM
```

See Also:

"Capture Process Overview"

Displaying SCN Values for Each Redo Log File Used by Each Capture Process

A sample query illustrates how to display information about the SCN values for archived redo log files that are registered for each capture process in a database.

This query displays the SCN values for these files for both local capture processes and downstream capture processes. This query also identifies redo log files that are no longer needed by any capture process at the local database.

The query displays the following information for each registered archived redo log file:

- The capture process name of a capture process that uses the file
- The name and location of the file at the local site
- The lowest SCN value for the information contained in the redo log file
- The lowest SCN value for the next redo log file in the sequence
- Whether the redo log file is purgeable

To display SCN values for each redo log file used by each capture process:

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 CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN NAME HEADING 'Archived Redo Log|File Name' FORMAT A25
COLUMN FIRST_SCN HEADING 'First SCN' FORMAT 99999999999
COLUMN NEXT_SCN HEADING 'Next SCN' FORMAT 99999999999
COLUMN PURGEABLE HEADING 'Purgeable?' FORMAT A10

SELECT r.CONSUMER_NAME,
       r.NAME,
       r.FIRST_SCN,
       r.NEXT_SCN,
       r.PURGEABLE
  FROM DBA_REGISTERED_ARCHIVED_LOG r, ALL_CAPTURE c
  WHERE r.CONSUMER_NAME = c.CAPTURE_NAME;
```

Your output looks similar to the following:

```
Capture
Process         Archived Redo Log
Name            File Name                    First SCN     Next SCN Purgeable?
--------------- ------------------------- ------------ ------------ ----------
CAP$_XOUT_1     /private1/ARCHIVE_LOGS/1_       509686       549100 YES
                3_502628294.dbf

CAP$_XOUT_1     /private1/ARCHIVE_LOGS/1_       549100       587296 YES
                4_502628294.dbf

CAP$_XOUT_1     /private1/ARCHIVE_LOGS/1_       587296       623107 NO
                5_502628294.dbf
```

The redo log files with `YES` for `Purgeable?` for all capture processes will never be needed by any capture process at the local database. These redo log files can be removed without affecting any existing capture process at the local database. The redo log files with `NO` for `Purgeable?` for one or more capture processes must be retained.

Listing the Parameter Settings for Each Capture Process

A sample query illustrates how to list the parameter settings for each capture process.

Capture process parameters determine how a capture process operates.

To list the parameter settings for each capture process:

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 CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A25
COLUMN PARAMETER HEADING 'Parameter' FORMAT A30
COLUMN VALUE HEADING 'Value' FORMAT A10
COLUMN SET_BY_USER HEADING 'Set by|User?' FORMAT A10

SELECT c.CAPTURE_NAME,
       PARAMETER,
       VALUE,
       SET_BY_USER
  FROM ALL_CAPTURE_PARAMETERS c, ALL_XSTREAM_OUTBOUND o
  WHERE c.CAPTURE_NAME=o.CAPTURE_NAME
  ORDER BY PARAMETER;
```

Your output looks similar to the following:

```
Capture
Process                                                                 Set by
Name                      Parameter                      Value      User?
------------------------- ------------------------------ ---------- ----------
CAP$_XOUT_1               CAPTURE_IDKEY_OBJECTS          N          NO
CAP$_XOUT_1               CAPTURE_SEQUENCE_NEXTVAL       N          NO
CAP$_XOUT_1               DISABLE_ON_LIMIT               N          NO
CAP$_XOUT_1               DOWNSTREAM_REAL_TIME_MINE      Y          NO
CAP$_XOUT_1               EXCLUDETAG                                NO
CAP$_XOUT_1               EXCLUDETRANS                              NO
CAP$_XOUT_1               EXCLUDEUSER                               NO
CAP$_XOUT_1               EXCLUDEUSERID                             NO
CAP$_XOUT_1               GETAPPLOPS                     Y          NO
CAP$_XOUT_1               GETREPLICATES                  N          NO
CAP$_XOUT_1               IGNORE_TRANSACTION                        NO
CAP$_XOUT_1               IGNORE_UNSUPPORTED_TABLE       *          NO
CAP$_XOUT_1               INCLUDE_OBJECTS                           NO
CAP$_XOUT_1               INLINE_LOB_OPTIMIZATION        N          NO
CAP$_XOUT_1               MAXIMUM_SCN                    INFINITE   NO
CAP$_XOUT_1               MAX_SGA_SIZE                   INFINITE   NO
CAP$_XOUT_1               MERGE_THRESHOLD                60         NO
CAP$_XOUT_1               MESSAGE_LIMIT                  INFINITE   NO
CAP$_XOUT_1               MESSAGE_TRACKING_FREQUENCY     2000000    NO
CAP$_XOUT_1               PARALLELISM                    0          NO
CAP$_XOUT_1               SKIP_AUTOFILTERED_TABLE_DDL    Y          NO
CAP$_XOUT_1               SPLIT_THRESHOLD                1800       NO
CAP$_XOUT_1               STARTUP_SECONDS                0          NO
CAP$_XOUT_1               TIME_LIMIT                     INFINITE   NO
CAP$_XOUT_1               TRACE_LEVEL                    0          NO
CAP$_XOUT_1               USE_RAC_SERVICE                N          NO
CAP$_XOUT_1               WRITE_ALERT_LOG                Y          NO
CAP$_XOUT_1               XOUT_CLIENT_EXISTS             Y          NO
```

Note:

If the `Set by User?` column is `NO` for a parameter, then the parameter is set to its default value. If the `Set by User?` column is `YES` for a parameter, then the parameter was set by a user and might or might not be set to its default value.

See Also:

- "XStream Out Process Subcomponents"
- "Setting a Capture Process Parameter"
- *Oracle Database PL/SQL Packages and Types Reference* for information about capture process parameters

Determining the Applied SCN for Each Capture Process

A sample query illustrates how to determine the applied SCN for each capture process.

The applied system change number (SCN) for a capture process is the SCN of the most recent logical change record (LCR) dequeued by the relevant outbound servers. All changes below this applied SCN have been processed by all outbound servers that process changes captured by the capture process.

To determine the applied SCN for each capture process:

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 CAPTURE_NAME HEADING 'Capture Process Name' FORMAT A30
   COLUMN APPLIED_SCN HEADING 'Applied SCN' FORMAT 99999999999

   SELECT CAPTURE_NAME, APPLIED_SCN FROM ALL_CAPTURE;
   ```

Your output looks similar to the following:

```
Capture Process Name            Applied SCN
------------------------------ ------------
CAP$_XOUT_1                          824825
```

Displaying the Redo Log Scanning Latency for Each Capture Process

A sample query illustrates how to display the redo log scanning latency for each capture process.

You can find the following information about each capture process by running the query in this section:

- The redo log scanning latency, which specifies the number of seconds between the creation time of the most recent redo log entry scanned by a capture process and the current time. This number might be relatively large immediately after you start a capture process.
- The seconds since last recorded status, which is the number of seconds since a capture process last recorded its status.
- The current capture process time, which is the latest time when the capture process recorded its status.
- The logical change record (LCR) creation time, which is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo data at the source database for the most recently captured LCR.

The information displayed by this query is valid only for an enabled capture process.

To display the redo log scanning latency for each capture process:

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 CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A12
COLUMN LATENCY_SECONDS HEADING 'Latency|in|Seconds' FORMAT 999999
COLUMN LAST_STATUS HEADING 'Seconds Since|Last Status' FORMAT 999999
COLUMN CAPTURE_TIME HEADING 'Current|Process|Time'
COLUMN CREATE_TIME HEADING 'Message|Creation Time' FORMAT 999999

SELECT CAPTURE_NAME,
       ((SYSDATE - CAPTURE_MESSAGE_CREATE_TIME)*86400) LATENCY_SECONDS,
       ((SYSDATE - CAPTURE_TIME)*86400) LAST_STATUS,
       TO_CHAR(CAPTURE_TIME, 'HH24:MI:SS MM/DD/YY') CAPTURE_TIME,
       TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_TIME
  FROM V$XSTREAM_CAPTURE;
```

Your output looks similar to the following:

```
Capture      Latency               Current
Process           in Seconds Since Process           Message
Name         Seconds   Last Status Time              Creation Time
------------ ------- ------------- ----------------- -----------------
CAP$_XOUT_1        1             1 10:32:52 02/28/11 10:32:52 02/28/11
```

The "`Latency in Seconds`" returned by this query is the difference between the current time (`SYSDATE`) and the "`Message Creation Time.`" The "`Seconds Since Last Status`" returned by this query is the difference between the current time (`SYSDATE`) and the "`Current Process Time.`"

Displaying the Extra Attributes Captured by a Capture Process

A sample query illustrates how to display the extra attributes captured by a capture process.

You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a capture process to capture one or more extra attributes and include the extra attributes in logical change records (LCRs).

To display extra attributes captured by a capture process:

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 CAPTURE_NAME HEADING 'Capture Process' FORMAT A20
COLUMN ATTRIBUTE_NAME HEADING 'Attribute Name' FORMAT A15
COLUMN INCLUDE HEADING 'Include Attribute in LCRs?' FORMAT A30

SELECT CAPTURE_NAME, ATTRIBUTE_NAME, INCLUDE
  FROM ALL_CAPTURE_EXTRA_ATTRIBUTES
  ORDER BY CAPTURE_NAME;
```

Your output looks similar to the following:

```
Capture Process      Attribute Name  Include Attribute in LCRs?
-------------------- --------------- ------------------------------
CAP$_XOUT_1          ROW_ID          NO
CAP$_XOUT_1          SERIAL#         NO
CAP$_XOUT_1          SESSION#        NO
CAP$_XOUT_1          THREAD#         NO
CAP$_XOUT_1          TX_NAME         YES
CAP$_XOUT_1          USERNAME        NO
```

Based on this output, the capture process named `xcapture` includes the transaction name (`tx_name`) in the LCRs that it captures, but this capture process does not include any other extra attributes in the LCRs that it captures.

> **See Also:**
>
> - "Extra Information in Row LCRs and DDL LCRs"
> - *Oracle Database PL/SQL Packages and Types Reference*for more information about the `INCLUDE_EXTRA_ATTRIBUTE` procedure

Monitoring XStream Rules

A sample query illustrates how to monitor XStream rules.

The `ALL_XSTREAM_RULES` view contains 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 'XStream|Component|Name' FORMAT A9
COLUMN STREAMS_TYPE HEADING 'XStream|Component|Type' FORMAT A9
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A13
COLUMN RULE_SET_TYPE HEADING 'Rule Set|Type' FORMAT A8
COLUMN STREAMS_RULE_TYPE HEADING '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 ALL_XSTREAM_RULES;
```

Your output looks similar to the following:

```
XStream   XStream
Component Component Rule          Rule Set Rule    Schema Object      Rule
Name      Type      Name          Type     Level   Name   Name        Type
```

```
--------- --------- ------------- -------- ------- ------ ----------- ----
XOUT      APPLY     ORDERS11      POSITIVE TABLE   OE     ORDERS      DML
XOUT      APPLY     ORDERS12      POSITIVE TABLE   OE     ORDERS      DDL
XOUT      APPLY     ORDER_ITEMS14 POSITIVE TABLE   OE     ORDER_ITEMS DML
XOUT      APPLY     ORDER_ITEMS15 POSITIVE TABLE   OE     ORDER_ITEMS DDL
XOUT      APPLY     HR16          POSITIVE SCHEMA  HR                 DML
XOUT      APPLY     HR17          POSITIVE SCHEMA  HR                 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 component by querying the `PURPOSE` column in the `ALL_APPLY` view.

The `ALL_XSTREAM_RULES` view contains more information about the rules used in an XStream configuration than what is shown in this example. For example, you can query this view to show information about the rule sets used by XStream components.

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_XSTREAM_RULES` view.

See Also:

Oracle Database Reference

Monitoring Declarative Rule-Based Transformations

A sample query illustrates how to monitor declarative rule-based transformations.

A declarative rule-based transformations is a rule-based transformation that covers one of a common set of transformation scenarios for row LCRs. Declarative rule-based transformations are run internally without using PL/SQL.

The query in this section displays the following information about each declarative rule-based transformation in a database:

- The owner of the rule for which a declarative rule-based transformation is specified.
- The name of the rule for which a declarative rule-based transformation is specified.
- The type of declarative rule-based transformation specified. The following types are possible: `ADD COLUMN`, `DELETE COLUMN`, `KEEP COLUMNS`, `RENAME COLUMN`, `RENAME SCHEMA`, and `RENAME TABLE`.
- The precedence of the declarative rule-based transformation. The precedence is the execution order of a transformation in relation to other transformations with the same step number specified for the same rule. For transformations with the same step number, the transformation with the lowest precedence is executed first.
- The step number of the declarative rule-based transformation. If more than one declarative rule-based transformation is specified for the same rule, then the transformation with the lowest step number is executed first. You can specify the step number for a declarative rule-based transformation when you create the transformation.

You must have `DBA` role in order to access the `DBA_XSTREAM_TRANSFORMATIONS` view.

Run the following query to display this information for the declarative rule-based transformations 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 RULE_OWNER HEADING 'Rule Owner' FORMAT A15
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A15
COLUMN DECLARATIVE_TYPE HEADING 'Declarative|Type' FORMAT A15
COLUMN PRECEDENCE HEADING 'Precedence' FORMAT 99999
COLUMN STEP_NUMBER HEADING 'Step Number' FORMAT 99999

SELECT RULE_OWNER,
       RULE_NAME,
       DECLARATIVE_TYPE,
       PRECEDENCE,
       STEP_NUMBER
  FROM DBA_XSTREAM_TRANSFORMATIONS
  WHERE TRANSFORM_TYPE = 'DECLARATIVE TRANSFORMATION';
```

Your output looks similar to the following:

```
                                Declarative
Rule Owner      Rule Name       Type            Precedence Step Number
--------------- --------------- --------------- ---------- -----------
XSTRMADMIN      JOBS26          RENAME TABLE             4           0
XSTRMADMIN      EMPLOYEES22     ADD COLUMN               3           0
```

Based on this output, the `ADD COLUMN` transformation executes before the `RENAME TABLE` transformation because the step number is the same (zero) for both transformations and the `ADD COLUMN` transformation has the lower precedence.

The `DBA_XSTREAM_TRANSFORMATIONS` view can display more detailed information about each transformation based on the declarative type of the transformation. Include a `WHERE` clause in the query with the `DECLARATIVE_TYPE` equal to the type of transformation, such as `ADD COLUMN`, `DELETE COLUMN`, and so on.

For example, the previous query listed an `ADD COLUMN` transformation and a `RENAME TABLE` transformation.

Note:

Precedence and step number pertain only to declarative rule-based transformations. They do not pertain to subset rule transformations or custom rule-based transformations.

- Displaying Information About ADD COLUMN Transformations
  A sample query illustrates how to display detailed information about the `ADD COLUMN` declarative rule-based transformations in a database.
- Displaying Information About RENAME TABLE Transformations
  A sample query illustrates how to display detailed information about the `RENAME TABLE` declarative rule-based transformations in a database.

See Also:

- "Rule-Based Transformations"
- "Managing Declarative Rule-Based Transformations"

Displaying Information About ADD COLUMN Transformations

A sample query illustrates how to display detailed information about the ADD COLUMN declarative rule-based transformations in a database.

You use the view DBA_XTREAM_TRANSFORMATIONS to display information about the columns that are added to row LCRs with the declarative rule-based transformation procedure DBMS_XSTREAM_ADM.

To display information about ADD COLUMN transformations:

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 RULE_OWNER HEADING 'Rule|Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A11
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN TABLE_NAME HEADING 'Table|Name' FORMAT A9
COLUMN COLUMN_NAME HEADING 'Column|Name' FORMAT A10
COLUMN COLUMN_VALUE HEADING 'Column|Value' FORMAT A10
COLUMN COLUMN_TYPE HEADING 'Column|Type' FORMAT A8

SELECT RULE_OWNER,
       RULE_NAME,
       SCHEMA_NAME,
       TABLE_NAME,
       COLUMN_NAME,
       ANYDATA.AccessDate(COLUMN_VALUE) "Value",
       COLUMN_TYPE
  FROM DBA_XSTREAM_TRANSFORMATIONS
  WHERE DECLARATIVE_TYPE = 'ADD COLUMN';
```

Your output looks similar to the following:

```
Rule       Rule        Schema Table     Column     Column     Column
Owner      Name        Name   Name      Name       Value      Type
---------- ----------- ------ --------- ---------- ---------- --------
XSTRMADMIN EMPLOYEES22 HR     EMPLOYEES BIRTH_DATE            SYS.DATE
```

This output show the following information about the ADD COLUMN declarative rule-based transformation:

- It is specified on the employees22 rule in the xstrmadmin schema.
- It adds a column to row LCRs that involve the employees table in the hr schema.
- The column name of the added column is BIRTH_DATE.
- The value of the added column is NULL. The COLUMN_VALUE column in the ALL_XSTREAM_TRANSFORMATIONS view is type ANYDATA. In this example, because the column

type is `DATE`, the `ANYDATA.AccessDate` member function is used to display the value. Use the appropriate member function to display values of other types.

- The column type of the added column is `DATE`.

Displaying Information About RENAME TABLE Transformations

A sample query illustrates how to display detailed information about the `RENAME TABLE` declarative rule-based transformations in a database.

You use the view `DBA_XSTREAM_TRANSFORMATIONS` to display information about declarative rule-based transformations that rename a table in a row logical change record (LCR).

To display information about `RENAME TABLE` transformations:

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 RULE_OWNER HEADING 'Rule|Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A10
COLUMN FROM_SCHEMA_NAME HEADING 'From|Schema|Name' FORMAT A10
COLUMN TO_SCHEMA_NAME HEADING 'To|Schema|Name' FORMAT A10
COLUMN FROM_TABLE_NAME HEADING 'From|Table|Name' FORMAT A15
COLUMN TO_TABLE_NAME HEADING 'To|Table|Name' FORMAT A15

SELECT RULE_OWNER,
       RULE_NAME,
       FROM_SCHEMA_NAME,
       TO_SCHEMA_NAME,
       FROM_TABLE_NAME,
       TO_TABLE_NAME
  FROM DBA_XSTREAM_TRANSFORMATIONS
  WHERE DECLARATIVE_TYPE = 'RENAME TABLE';
```

Your output looks similar to the following:

```
                      From       To         From            To
Rule       Rule       Schema     Schema     Table           Table
Owner      Name       Name       Name       Name            Name
---------- ---------- ---------- ---------- --------------- ---------------
XSTRMADMIN JOBS26     HR         HR         HR.JOBS         HR.ASSIGNMENTS
```

This output show the following information about the `RENAME TABLE` declarative rule-based transformation:

- It is specified on the `jobs26` rule in the `xstrmadmin` schema.
- It renames the `hr.jobs` table in row LCRs to the `hr.assignments` table.

7

Troubleshooting XStream Out

You can diagnose and correct problems with an XStream Out configuration.

- Diagnosing Problems with XStream Out
  You can diagnose problems with XStream Out by using several different techniques.
- Problems and Solutions for XStream Out
  You can implement solutions for common problems with XStream Out.
- How to Get More Help with XStream Out
  Oracle Support can provide more help with XStream Out.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- "Configuring XStream Out"

Diagnosing Problems with XStream Out

You can diagnose problems with XStream Out by using several different techniques.

- Viewing Alerts
  An **alert** is a warning about a potential problem or an indication that a critical threshold has been crossed.
- Checking the Trace File and Alert Log for Problems
  Messages about each capture process and outbound server are recorded in trace files for the database in which the process is running.

Viewing Alerts

An **alert** is a warning about a potential problem or an indication that a critical threshold has been crossed.

There are two types of alerts:

- **Stateless:** Alerts that indicate single events that are not necessarily tied to the system state. For example, an alert that indicates that a capture aborted with a specific error is a stateless alert.
- **Stateful:** Alerts that are associated with a specific system state. Stateful alerts are usually based on a numeric value, with thresholds defined at warning and critical levels. For example, an alert on the current Streams pool memory usage percentage, with the warning level at 85% and the critical level at 95%, is a stateful alert.

An Oracle database generates a stateless alert under the following conditions:

- A capture process aborts.
- An outbound server aborts.

An Oracle database generates a stateful XStream alert when the Streams pool memory usage exceeds the percentage specified by the `STREAMS_POOL_USED_PCT` metric. You can manage this metric with the `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package.

You can view alerts in Oracle Enterprise Manager Cloud Control, or you can query the following data dictionary views:

- The `DBA_OUTSTANDING_ALERTS` view records current stateful alerts. The `DBA_ALERT_HISTORY` view records stateless alerts and stateful alerts that have been cleared. For example, if the memory usage in the Streams pool exceeds the specified threshold, then a stateful alert is recorded in the `DBA_OUTSTANDING_ALERTS` view.
- The `DBA_ALERT_HISTORY` data dictionary view shows alerts that have been cleared from the `DBA_OUTSTANDING_ALERTS` view. For example, if the memory usage in the streams pool falls below the specified threshold, then the alert recorded in the `DBA_OUTSTANDING_ALERTS` view is cleared and moved to the `DBA_ALERT_HISTORY` view.

For example, to list the current stateful alerts, run the following query on the `DBA_OUTSTANDING_ALERTS` view:

```
COLUMN REASON HEADING 'Reason for Alert' FORMAT A35
COLUMN SUGGESTED_ACTION HEADING 'Suggested Response' FORMAT A35

SELECT REASON, SUGGESTED_ACTION
   FROM DBA_OUTSTANDING_ALERTS
   WHERE MODULE_ID LIKE '%XSTREAM%';
```

To list the stateless alerts and cleared XStream stateful alerts, run the following query on the `DBA_ALERT_HISTORY` view:

```
COLUMN REASON HEADING 'Reason for Alert' FORMAT A35
COLUMN SUGGESTED_ACTION HEADING 'Suggested Response' FORMAT A35

SELECT REASON, SUGGESTED_ACTION
   FROM DBA_ALERT_HISTORY
   WHERE MODULE_ID LIKE '%XSTREAM%';
```

Most alerts are cleared automatically when the cause of the problem disappears or is acknowledged by the database administrator.

See Also:

- *Oracle Database Administrator’s Guide* for information about alerts and for information about subscribing to the `ALERT_QUE` queue to receive notifications when new alerts are generated
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SERVER_ALERT` package
- "Configure the Streams pool"
- *Oracle Database Get Started with Performance Tuning* for more information on clearing and purging alerts with Oracle Enterprise Manager Cloud Control

Checking the Trace File and Alert Log for Problems

Messages about each capture process and outbound server are recorded in trace files for the database in which the process is running.

A local capture process runs on a source database and a downstream capture process runs on a downstream database. These trace file messages can help you to identify and resolve problems in an XStream Out configuration.

All trace files for background processes are written to the Automatic Diagnostic Repository. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file.

For example, on some operating systems, the trace file name for a process is `*sid*_*xxxx*_*iiiii*.trc`, where:

- *`sid`* is the system identifier for the database
- *`xxxx`* is the name of the process
- *`iiiii`* is the operating system process number

Also, you can set the `write_alert_log` parameter to `y` for both a capture process and an outbound server. When this parameter is set to `y`, which is the default setting, the alert log for the database contains messages about why the capture process or outbound server stopped.

You can control the information in the trace files by setting the `trace_level` capture process or outbound server apply parameter using the `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package.

- Capture Process Trace Files
  A capture process is an Oracle background process named `CP`*`nn`*, where *`nn`* can include letters and numbers.
- Logminer Trace Files
  Logminer trace files are useful in understanding issues with XStream Out.
- Outbound Server Trace File
  An outbound server is an Oracle background process named `AP`*`nn`*, where *`nn`* can include letters and numbers.
- Client Application Trace Files
  Client application trace files can help to isolate a problem with XStream Out.

See Also:

- *Oracle Database Administrator's Guide*for more information about trace files and the alert log, and for more information about their names and locations
- *Oracle Database PL/SQL Packages and Types Reference*for more information about setting the `trace_level` capture process parameter and the `trace_level` apply parameter
- Your operating system specific Oracle documentation for more information about the names and locations of trace files

Capture Process Trace Files

A capture process is an Oracle background process named `CP`*`nn`*, where *`nn`* can include letters and numbers.

For example, on some operating systems, if the system identifier for a database running a capture process is `hqdb` and the capture process number is `01`, then the trace file for the capture process starts with `hqdb_CP01`.

See Also:

"Displaying Change Capture Information About Each Capture Process" for a query that displays the capture process number of a capture process

Logminer Trace Files

Logminer trace files are useful in understanding issues with XStream Out.

The logminer trace files are created when the parallelism capture process parameter is set to a value greater than 0. There are at least 3 logminer trace files that are generated and written to the Automated Diagnostic Repository.

Outbound Server Trace File

An outbound server is an Oracle background process named `AP`*`nn`*, where *`nn`* can include letters and numbers.

For example, on some operating systems, if the system identifier for a database running an outbound server is `hqdb` and the outbound server number is `01`, then the trace file for the outbound server starts with `hqdb_ap01_xxxx.trc`.

An outbound server also uses other processes. Information about an outbound server might be recorded in the trace file for one or more of these processes. The process name of the reader server and apply servers is `AS`*`nn`*, where *`nn`* can include letters and numbers. So, on some operating systems, if the system identifier for a database running an outbound server is `hqdb` and the process number is `01`, then the trace file that contains information about a process used by an outbound server starts with `hqdb_AS01`.

See Also:

"Monitoring Session Information About XStream Out Components"

Client Application Trace Files

Client application trace files can help to isolate a problem with XStream Out.

When troubleshooting errors, isolating a problem to a key component, or identifying potential performance issues, it is a good idea to examine the trace files from all of the key sources in your XStream environment. One key source to check is the client application trace files. The client trace files are located in the directory: `$ORACLE_HOME/diag/clients/`.

Problems and Solutions for XStream Out

You can implement solutions for common problems with XStream Out.

In general, you can troubleshoot XStream outbound servers in the same way that you troubleshoot Oracle 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.

- 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.
- 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 Capture Process Is Missing Required Redo Log Files
  When a capture process is started or stopped and restarted, it might need to scan redo log files that were generated before the log file that contains the SCN that corresponds to the required checkpoint SCN, and these files might have been removed.
- 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.
- The XStream Out Client Application Is Unresponsive
  The XStream client application in an XStream Out configuration is unresponsive.

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.

To display information about the XStream Out servers that are accessible to the connect user:

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 to determine the connect user:

```
SELECT SERVER_NAME,
       CONNECT_USER,
       CAPTURE_NAME,
       SOURCE_DATABASE,
       CAPTURE_USER,
       QUEUE_OWNER
  FROM ALL_XSTREAM_OUTBOUND;
```

This query displays the name of the user (`connect_user`) who can connect to the outbound server and process the outbound LCRs.

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 Out and Security"
- *Oracle Call Interface Developer's Guide*

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 STATE HEADING 'State' FORMAT A15
COLUMN CREATE_MESSAGE HEADING 'Last LCR|Create Time'
COLUMN ENQUEUE_MESSAGE HEADING 'Last|Enqueue Time'

SELECT CAPTURE_NAME, STATE,
       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$XSTREAM_CAPTURE;
```

This query shows the current state of the capture process. This query also 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:

Oracle Database Reference

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.

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

  To disable LCR tracking when you use this method, set the `message_tracking_frequency` capture process parameter to `NULL` or exit the session.
- Running the `SET_MESSAGE_TRACKING` procedure in the `DBMS_XSTREAM_ADM` package

To disable LCR tracking when you use this method, set the `tracking_label` parameter to `NULL` in the `SET_MESSAGE_TRACKING` procedure or exit the session.

After using one of these methods, use the `V$XSTREAM_MESSAGE_TRACKING` view to monitor the progress of LCRs through a stream. By tracking an LCR through the stream, you can determine where the LCR is blocked.

In addition, if a propagation is used to send LCRs in the configuration, then you can check the current state of the propagation sender by running the following query:

```
SELECT STATE FROM V$PROPAGATION_SENDER;
```

You can check the current state of an outbound server by running the following query:

```
SELECT SERVER_NAME, STATE FROM V$XSTREAM_OUTBOUND_SERVER;
```

You can verify that the client application is attached to the outbound server by running the following query:

```
COLUMN SERVER_NAME HEADING 'Capture|Name' FORMAT A30
COLUMN STATUS HEADING 'Status' FORMAT A8

SELECT SERVER_NAME, STATUS FROM ALL_XSTREAM_OUTBOUND;
```

The `STATUS` column shows `ATTACHED` when the client application is attached to the outbound server.

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 Database PL/SQL Packages and Types Reference for information about the `message_tracking_frequency` capture process parameter

The Capture Process Is Missing Required Redo Log Files

When a capture process is started or stopped and restarted, it might need to scan redo log files that were generated before the log file that contains the SCN that corresponds to the required checkpoint SCN, and these files might have been removed.

You can query the `ALL_CAPTURE` data dictionary view to determine the required checkpoint SCN for a capture process. It is also helpful to query the `V$XSTREAM_CAPTURE` and check the `STATE` column. The state of a capture process describes what the capture process is doing currently. In this case, you can gain additional insight as to why the capture process is missing or waiting for redo log files.

```
COLUMN CAPTURE_NAME HEADING 'Capture Name' FORMAT A30
COLUMN STATE HEADING 'State' FORMAT A30

SELECT CAPTURE_NAME, STATE FROM V$XSTREAM_CAPTURE;
```

```
CAPTURE_NAME             STATE
------------------       -----------------
XOUT_SRC_CAPTURE         WAITING FOR REDO
```

Additional information might be displayed along with the state information when you query the `V$XSTREAM_CAPTURE` view. The additional information can help you to determine why the capture process is waiting for redo. For example, a statement similar to the following might appear for the `STATE` column when you query the view:

```
WAITING FOR REDO: LAST SCN MINED 6700345
```

In this case, the output shows the last system change number (SCN) scanned by the capture process. In other cases, the output might display the redo log file name explicitly. Either way, the additional information can help you identify the redo log file for which the capture process is waiting. To correct the problem, make any missing redo log files available to the capture process.

Problem: Required Redo Log Files Were Removed

Removing required redo log files before they are scanned by a capture process causes the capture process to abort and results in the following error in a capture process trace file:

```
ORA-01291: missing logfile
```

Solution: Restore Missing Redo Log Files and Prevent Future Problems

If you see this error, then try restoring any missing redo log files and restarting the capture process. You can check the `V$LOGMNR_LOGS` dynamic performance view to determine the missing SCN range, and add the relevant redo log files. A capture process needs the redo log file that includes the required checkpoint SCN and all subsequent redo log files. You can query the `REQUIRED_CHECKPOINT_SCN` column in the `ALL_CAPTURE` data dictionary view to determine the required checkpoint SCN for a capture process.

If the capture process is disabled for longer than the amount of time specified in the `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter, then information about the missing redo log files might have been replaced in the control file. You can query the `V$ARCHIVE_LOG` view to see if the log file names are listed. If they are not listed, then you can register them with a `ALTER DATABASE REGISTER OR REPLACE LOGFILE` SQL statement.

If you are using the fast recovery area feature of Recovery Manager (RMAN) on a source database in an XStream environment, then RMAN might delete archived redo log files that are required by a capture process. RMAN might delete these files when the disk space used by the recovery-related files is nearing the specified disk quota for the fast recovery area. To prevent this problem in the future, complete one or more of the following actions:

- Increase the disk quota for the fast recovery area. Increasing the disk quota makes it less likely that RMAN will delete a required archived redo log file, but it will not always prevent the problem.
- Configure the source database to store archived redo log files in a location other than the fast recovery area. A local capture process will be able to use the log files in the other location if the required log files are missing in the fast recovery area. In this case, a database administrator must manage the log files manually in the other location.

RMAN always ensures that archived redo log files are backed up before it deletes them. If RMAN deletes an archived redo log file that is required by a capture process, then RMAN records this action in the alert log.

See Also:

- "Capture Processes"
- "XStream Out and Recovery Manager"
- "Displaying Redo Log Files That Are Required by Each Capture Process"

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 ALL_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;
/
```

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.

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:

- *Oracle Database Reference*
- *Oracle Database Administrator's Guide* for information about setting initialization parameters

Problem 2: The Maximum SGA Size for the Capture Process Is Too Small

The `max_sga_size` capture process parameter controls the amount of system global area (SGA) memory allocated specifically to the capture process, in megabytes.

To determine whether the maximum SGA size for the capture process is too small:

1. Connect to the database running the XStream component 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:

   - Query the `V$XSTREAM_CAPTURE` view:

     ```
     SELECT CAPTURE_NAME AS CAP,
             SGA_USED/(1024*1024) AS USED,
             SGA_ALLOCATED/(1024*1024) AS ALLOCATED,
             TOTAL_MESSAGES_CAPTURED AS CAPTURED,
             TOTAL_MESSAGES_ENQUEUED AS ENQUEUED
       FROM V$XSTREAM_CAPTURE;
     ```

     If the `USED` field is equal to or almost equal to the `ALLOCATED` field in the output, then you might need to increase the maximum SGA size for the capture process.

   - Query the `V$LOGMNR_SESSION` view:

     ```
     SELECT SESSION_NAME AS CAP,
             MAX_MEMORY_SIZE/(1024*1024) AS LMMAX,
             USED_MEMORY_SIZE/(1024*1024) AS LMUSED,
             USED_MEMORY_SIZE/MAX_MEMORY_SIZE AS PCT
       FROM V$LOGMNR_SESSION;
     ```

     If the `PCT` field is equal to or almost equal to 1 in the output, then you might need to increase the maximum SGA size for the capture process.

Solution 2

To correct problem 2:

- Increase the maximum SGA size for the capture process by modifying the `max_sga_size` capture process parameter.

See Also:

"Setting a Capture Process Parameter"

Problem 3: Programming Errors

If there is enough memory in the Streams pool and the `MAX_SGA_SIZE` capture process parameter and apply parameter are set correctly, then check your client application for programming errors.

Solution 3

To correct problem 3:

- Correct the programming errors.

How to Get More Help with XStream Out

Oracle Support can provide more help with XStream Out.

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 In

You can configure and manage an XStream In environment.

- XStream In Concepts
  Become familiar with the concepts related to XStream In.
- Configuring XStream In
  You can configure the Oracle Database components that are used by XStream.
- Managing XStream In
  You can manage an XStream In configuration.
- Monitoring XStream In
  You can monitor an XStream In configuration by querying data dictionary views.
- Troubleshooting XStream In
  You can diagnose and correct problems with an XStream In configuration.

8

XStream In Concepts

Become familiar with the concepts related to XStream In.

- Introduction to 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.
- The Inbound Server
  With XStream In, an inbound server receives database changes from a client application.
- Position of LCRs and XStream In
  A client application streams LCRs to an XStream In inbound server.
- XStream In and Performance Considerations
  There are considerations for XStream In and performance.
- XStream In and Security
  Understand security related to the client application and XStream components, as well as the privileges required by the apply user for an inbound server.
- XStream In and Other Oracle Database Components
  XStream In can work with other Oracle Database components.

Introduction to 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 Oracle Replication features:

- High performance processing of DML changes, optionally with 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 Replication, including LOBs, `LONG`, `LONG RAW`, `JSON`, `BOOLEAN`, 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.

See Also:

- *Oracle Call Interface Developer's Guide*
- *Oracle Database XStream Java API Reference*

The Inbound Server

With XStream In, an inbound server receives database changes from a client application.

- Overview of Inbound Servers
  An **inbound server** is an optional Oracle background process that receives LCRs from a client application.
- Data Types Applied by Inbound Servers
  An inbound server supports most data types.
- LCR Processing Options for Inbound Servers
  An inbound server can either apply LCRs directly or send LCRs to an apply handler for processing. Your options for LCR processing depend on whether the LCR received by an inbound server is a row LCR or a DDL LCR.
- Inbound Servers and RESTRICTED SESSION
  Enabling and disabling restricted session affects inbound servers.
- Inbound Server Components
  An inbound server consists of the following subcomponents: a reader server, a coordinator process, and one or more apply servers.
- Considerations for Inbound Servers
  There are several considerations for inbound servers.
- The Error Queue for an Inbound Server
  The error queue contains all of the current apply errors for a database. If there are multiple inbound servers in a database, then the error queue contains the apply errors for each inbound server.

Overview of Inbound Servers

An **inbound server** is an optional Oracle background process that receives LCRs from a client application.

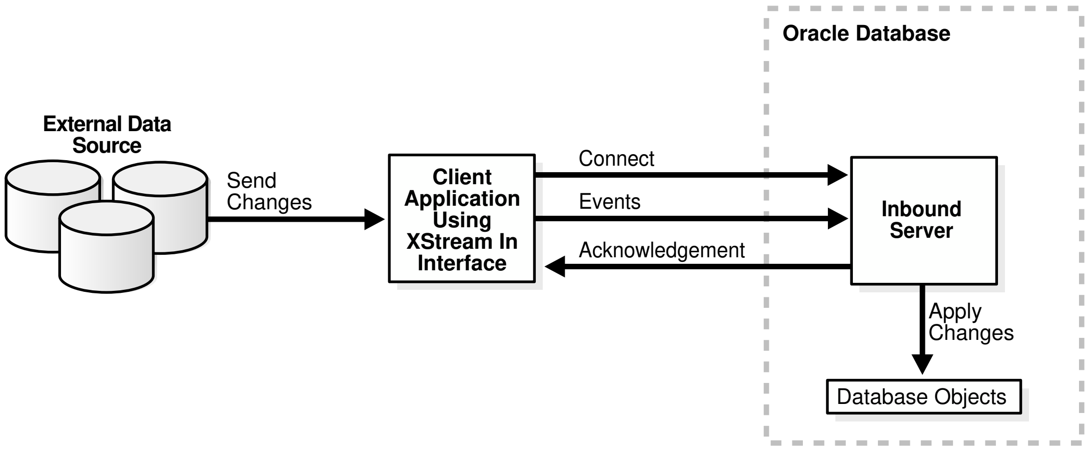
Specifically, a client application can attach to an inbound server and send row changes, DDL changes, and procedure calls encapsulated in LCRs.

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 8-1 shows an inbound server configuration.

Figure 8-1 XStream In Inbound Server



Note:

An inbound server uses a queue that is not shown in Figure 8-1. An inbound server's queue is only used to store error transactions.

Data Types Applied by Inbound Servers

An inbound server supports most data types.

When applying row LCRs resulting from DML changes to tables, an inbound server applies changes made to columns of the following data types:

- `JSON`
- `BOOLEAN`
- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `FLOAT`
- `LONG`
- `DATE`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`

- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `LONG RAW`
- `UROWID`
- `CHAR`
- `NCHAR`
- `CLOB` with `BASICFILE` or `SECUREFILE` storage
- `NCLOB` with `BASICFILE` or `SECUREFILE` storage
- `BLOB` with `BASICFILE` or `SECUREFILE` storage
- `XMLType` stored as `CLOB`, object relational, or as binary XML
- Object types
- Varrays
- `REF` data types
- The following Oracle-supplied types: `ANYDATA`, `SDO_GEOMETRY`, and media types
- `BFILE`

If XStream is replicating data for an object type, then the replication must be unidirectional, and all replication sites must agree on the names and data types of the attributes in the object type. You establish the names and data types of the attributes when you create the object type. For example, consider the following object type:

```
CREATE TYPE cust_address_typ AS OBJECT
       (street_address     VARCHAR2(40),
        postal_code        VARCHAR2(10),
        city               VARCHAR2(30),
        state_province     VARCHAR2(10),
        country_id         CHAR(2));
/
```

At all replication sites, `street_address` must be `VARCHAR2(40)`, `postal_code` must be `VARCHAR2(10)`, `city` must be `VARCHAR2(30)`, and so on.

Note:

- The maximum size of the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types has been increased in Oracle Database 12*c* when the `COMPATIBLE` initialization parameter is set to `12.0.0` and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`.
- `XMLType` stored as a `CLOB` is deprecated in Oracle Database 12*c* Release 1 (12.1).
- For `BFILE`, only the data type structure is replicated and not the content of the `BFILE` that exists on the file system.

See Also:

Oracle Database SQL Language Reference for information about data types

LCR Processing Options for Inbound Servers

An inbound server can either apply LCRs directly or send LCRs to an apply handler for processing. Your options for LCR processing depend on whether the LCR received by an inbound server is a row LCR or a DDL LCR.

By default, an inbound server applies LCRs directly. The inbound server executes the change in the LCR on the database object identified in the LCR. The inbound server either successfully applies the change in the LCR or, if a conflict or an apply error is encountered, tries to resolve the error with a conflict handler or a user-specified procedure called an error handler.

If a conflict handler can resolve the conflict, then it either applies the LCR or it discards the change in the LCR. If an error handler can resolve the error, then it should apply the LCR, if appropriate. An error handler can resolve an error by modifying the LCR before applying it. If the conflict handler or error handler cannot resolve the error, then the inbound server places the transaction, and all LCRs associated with the transaction, into the error queue.

Instead of applying LCRs directly, you can process LCRs in a customized way with apply handlers. When you use an apply handler, an inbound server passes an LCR to a collection of SQL statements or to a user-defined PL/SQL procedure for processing. An apply handler can process the LCR in a customized way.

There are several types of apply handlers. This section uses the following categories to describe apply handlers:

Table 8-1 Characteristics of Apply Handlers

Category	Description
Mechanism	The means by which the apply handler processes LCRs. The mechanism for an apply handler is either SQL statements or a user-defined PL/SQL procedure.
Type of LCR	The type of LCR processed by the apply handler. The LCR type is either row LCR, DDL LCR, or transaction control directive.
Scope	The level at which the apply handler is set. The scope is either one operation on one table or all operations on all database objects.
Number allowed for each inbound server	The number of apply handlers of a specific type allowed for each inbound server. The number allowed is either one or many.

- Procedure DML Handlers
  A **procedure DML handler** uses a PL/SQL procedure to process row LCRs.
- Error Handlers
  An error handler is similar to a procedure DML handler. The difference between the two is that an error handler is invoked only if an apply error results when an inbound server tries to apply a row LCR for the specified operation on the specified table.
- DDL Handlers
  A **DDL handler** uses a PL/SQL procedure to process DDL LCRs.

- Precommit Handlers
  A **precommit handler** uses a PL/SQL procedure to process commit directive for transactions that include row LCRs.

Procedure DML Handlers

A **procedure DML handler** uses a PL/SQL procedure to process row LCRs.

A procedure DML handler has the following characteristics:

- Mechanism: A user-defined PL/SQL procedure
- Type of LCR: Row LCR
- Scope: One operation on one table
- Number allowed for each inbound server: Many, but only one can be specified for the same operation on the same table

For each table associated with an inbound server, you can set a separate procedure DML handler to process each of the following types of operations in row LCRs:

- `INSERT`
- `UPDATE`
- `DELETE`
- `LOB_UPDATE`

A procedure DML handler is invoked when the inbound server receives a row LCR that performs the specified operation on the specified table. For example, the `hr.employees` table can have one procedure DML handler to process `INSERT` operations and a different procedure DML handler to process `UPDATE` operations. Alternatively, the `hr.employees` table can use the same procedure DML handler for each type of operation.

The PL/SQL procedure can perform any customized processing of row LCRs. For example, if you want each insert into a particular table at the source database to result in inserts into multiple tables at the destination database, then you can create a user-defined PL/SQL procedure that processes `INSERT` operations on the table to accomplish this. Procedure DML handlers can modify the column values in row LCRs.

Error Handlers

An error handler is similar to a procedure DML handler. The difference between the two is that an error handler is invoked only if an apply error results when an inbound server tries to apply a row LCR for the specified operation on the specified table.

An **error handler** has the following characteristics:

- Mechanism: A user-defined PL/SQL procedure
- Type of LCR: Row LCR
- Scope: One operation on one table
- Number allowed for each inbound server: Many, but only one can be specified for the same operation on the same table

See Also:

"Procedure DML Handlers"

DDL Handlers

A **DDL handler** uses a PL/SQL procedure to process DDL LCRs.

A DDL handler has the following characteristics:

- Mechanism: A user-defined PL/SQL procedure
- Type of LCR: DDL LCR
- Scope: All DDL LCRs received by the inbound server
- Number allowed for each inbound server: One

The user-defined PL/SQL procedure can perform any customized processing of DDL LCRs. For example, to log DDL changes before applying them, you can create a procedure that processes DDL operations to accomplish this.

Precommit Handlers

A **precommit handler** uses a PL/SQL procedure to process commit directive for transactions that include row LCRs.

A precommit handler has the following characteristics:

- Mechanism: A user-defined PL/SQL procedure
- Type of LCR: Commit directive for transactions that include row LCRs
- Scope: All row LCRs with commit directives received by the inbound server
- Number allowed for each inbound server: One

You can use a precommit handler to audit commit directives for LCRs. A commit directive is a transaction control directive that contains a `COMMIT`. A precommit handler is a user-defined PL/SQL procedure that can receive the commit information for a transaction and process the commit information in any customized way. A precommit handler works with a procedure DML handler.

For example, a precommit handler can improve performance by caching data for the length of a transaction. This data can include cursors, temporary LOBs, data from a message, and so on. The precommit handler can release or execute the objects cached by the handler when a transaction completes.

Inbound Servers and RESTRICTED SESSION

Enabling and disabling restricted session affects inbound servers.

When restricted session is enabled during system startup by issuing a `STARTUP RESTRICT` statement, inbound servers do not start, even if they were running when the database shut down. When the restricted session is disabled, each inbound server that was not stopped is started.

When restricted session is enabled in a running database by the SQL statement `ALTER SYSTEM ENABLE RESTRICTED SESSION`, it does not affect any running inbound servers. These inbound servers continue to run and send LCRs to an XStream client application. If a stopped inbound server is started in a restricted session, then the inbound server does not actually start until the restricted session is disabled.

Inbound Server Components

An inbound server consists of the following subcomponents: a reader server, a coordinator process, and one or more apply servers.

An inbound server consists of the following subcomponents:

- A **reader server** that receives LCRs from an XStream client application. The reader server is a process that computes dependencies between logical change records (LCRs) and assembles LCRs into transactions. The reader server then returns the assembled transactions to the coordinator process.

  You can view the state of the reader server for an inbound server by querying the `V$XSTREAM_APPLY_READER` dynamic performance view.

- A **coordinator process** that gets transactions from the reader server and passes them to apply servers. The coordinator process name is `AP`*`nn`*, where *`nn`* can include letters and numbers. The coordinator process is an Oracle background process.

  You can view the state of a coordinator process by querying the `V$XSTREAM_APPLY_COORDINATOR` dynamic performance view.

- One or more **apply servers** that apply LCRs to database objects as DML or DDL statements or that pass the LCRs to their appropriate apply handlers. Apply servers can also enqueue LCRs into the persistent queue portion of a queue specified by the `DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION` procedure. Each apply server is a process. If an apply server encounters an error, then it then tries to resolve the error with a user-specified conflict handler or error handler. If an apply server cannot resolve an error, then it rolls back the transaction and places the entire transaction, including all of its LCRs, in the error queue.

  When an apply server commits a completed transaction, this transaction has been applied. When an apply server places a transaction in the error queue and commits, this transaction also has been applied.

  You can view the state of each apply server for an inbound server by querying the `V$XSTREAM_APPLY_SERVER` dynamic performance view.

The reader server and the apply server process names are `AS`*`nn`*, where *`nn`* can include letters and numbers. If a transaction being handled by an apply server has a dependency on another transaction that is not known to have been applied, then the apply server contacts the coordinator process and waits for instructions. The coordinator process monitors all of the apply servers to ensure that transactions are applied and committed in the correct order.

See Also:

- *Oracle Database Reference* for more information about `V$XSTREAM_APPLY_READER` dynamic performance view
- *Oracle Database Reference* for more information about `V$XSTREAM_APPLY_COORDINATOR` dynamic performance view
- *Oracle Database Reference* for more information about `V$XSTREAM_APPLY_SERVER` dynamic performance view

Considerations for Inbound Servers

There are several considerations for 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 inbound server 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 inbound server. A trigger's firing property works the same for apply processes and inbound servers.
- An inbound server ignores the setting for the `ignore_transaction` apply parameter because LCRs sent to the inbound server by the client application might not have transaction ID values.
- An inbound server ignores the setting for the `maximum_scn` apply parameter because LCRs sent to the inbound server by the client application might not have SCN values.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about apply parameters

The Error Queue for an Inbound Server

The error queue contains all of the current apply errors for a database. If there are multiple inbound servers in a database, then the error queue contains the apply errors for each inbound server.

Trusted users can view apply errors by querying the `DBA_APPLY_ERROR` data dictionary view or by using Oracle Enterprise Manager Cloud Control. The `DBA_APPLY_ERROR` data dictionary view enables the trusted user to see information about apply errors for other users. Untrusted users can view apply errors by querying the `ALL_APPLY_ERROR` data dictionary view. This view shows only apply errors for the untrusted user.

Also, trusted users can view more detailed information about apply errors by querying the `DBA_APPLY_ERROR_MESSAGES` data dictionary view. Untrusted users can view more detailed information about apply errors by querying the `ALL_APPLY_ERROR_MESSAGES` data dictionary view. These views include information about the row that caused the error in an error transaction.

The error queue stores information about transactions that could not be applied successfully by the inbound server running in a database. A transaction can include many LCRs. When an unhandled error occurs during apply, an inbound server automatically moves all of the LCRs in the transaction that satisfy the inbound server's rule sets to the error queue.

You can correct the condition that caused an error and then reexecute the transaction that caused the error. For example, you might modify a row in a table to correct the condition that caused an error.

When the condition that caused the error has been corrected, you can either reexecute the transaction in the error queue using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure, or you can delete the transaction from the error queue using the `DELETE_ERROR` or `DELETE_ALL_ERRORS` procedure. These procedures are in the `DBMS_APPLY_ADM` package.

When you reexecute a transaction in the error queue, you can specify that the transaction be executed either by the user who originally placed the error in the error queue or by the user who is reexecuting the transaction. Also, the current tag for the inbound server is used when you reexecute a transaction in the error queue.

A reexecuted transaction uses any relevant apply handlers and conflict resolution handlers. If, to resolve the error, a row LCR in an error queue must be modified before it is executed, then you can configure a procedure DML handler to process the row LCR that caused the error in the error queue. In this case, the DML handler can modify the row LCR to avoid a repetition of the same error. The row LCR is passed to the DML handler when you reexecute the error containing the row LCR. For example, a procedure DML handler might modify one or more columns in the row LCR to avoid a repetition of the same error.

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 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 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.

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 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.

Example 8-1 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
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

See Also:

- "Position Order in an LCR Stream"
- "Displaying the Position Information for an Inbound Server"

XStream In and Performance Considerations

There are considerations for XStream In and performance.

- Optimizing XStream In Performance for Large Transactions
  For small transactions, XStream In does not begin to apply the logical change records (LCRs) until the inbound server receives a commit LCR for the transaction from the source. As a performance optimization, an inbound server can use eager apply to begin to apply large transactions before it receives the commit LCR.

- Optimizing Transaction Apply Scheduling
  When the constraints on the target tables match the constraints on the source tables, you can optimize dependency computation by setting the `compute_lcr_dep_on_arrival` apply parameter for an inbound server to `Y`.

Optimizing XStream In Performance for Large Transactions

For small transactions, XStream In does not begin to apply the logical change records (LCRs) until the inbound server receives a commit LCR for the transaction from the source. As a performance optimization, an inbound server can use eager apply to begin to apply large transactions before it receives the commit LCR.

The `eager_size` apply parameter controls the minimum number of LCRs received by the inbound server before eager apply begins. When the number of LCRs in a transaction exceeds the value of the `eager_size` apply parameter, the inbound server begins to apply the LCRs. The default value for this parameter is 9500. You can modify the parameter value to optimize XStream In performance in your environment.

Large transactions may require additional apply servers to apply the LCRs. After eager apply starts for a transaction, an inbound server can automatically create additional apply servers to apply the LCRs. The `max_parallelism` apply parameter controls the maximum number of apply servers for an inbound server.

If an inbound server automatically creates additional apply servers, and some of them are idle for a period of time, then XStream In determines that they are no longer necessary and removes them automatically. However, the number of apply servers never goes below the value specified by the `parallelism` apply parameter. Any statistics for these apply servers are aggregated as apply server 0 (zero).

For an inbound server to use eager apply for large transactions, the value of the `eager_size` apply parameter must be less than the value of the `txn_lcr_spill_threshold` apply parameter. When the value of `txn_lcr_spill_threshold` is lower than `eager_size`, a transaction spills to disk before eager apply begins, and a an inbound server cannot use eager apply for a transaction that has spilled to disk.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference*
- "Managing Eager Errors Encountered by an Inbound Server"

Optimizing Transaction Apply Scheduling

When the constraints on the target tables match the constraints on the source tables, you can optimize dependency computation by setting the `compute_lcr_dep_on_arrival` apply parameter for an inbound server to `Y`.

If the constraints do not match, then set this apply parameter to `N`, the default.

If this apply parameter is set to `Y`, then the dependencies are computed as the LCRs for the transaction are received. If this apply parameter is set to `N`, then the dependencies are computed only after all the LCRs for a committed transaction are received.

Regardless of `compute_lcr_dep_on_arrival` apply parameter setting, the before image of the key columns must be available in the LCRs received by the inbound server. Key columns include primary key columns, foreign key column, and unique constraint columns. In an XStream configuration in which an inbound server applies changes captured by a capture process in an XStream Out configuration, supplemental logging ensures that the required information is in the LCRs.

See Also:

- "If Required, Configure Supplemental Logging"
- *Oracle Database PL/SQL Packages and Types Reference*

XStream In and Security

Understand security related to the client application and XStream components, as well as the privileges required by the apply user for an inbound server.

- The XStream In Client Application and Security
  XStream In allows an application to send LCRs to an inbound server, and an inbound server can apply the database changes in the LCRs to the database.
- XStream In Component-Level Security
  All the components of the XStream In configuration run as the same user. This user is the apply user for the inbound server.
- Privileges Required by the Apply User for an Inbound Server
  An inbound server applies LCRs in the security domain of its apply user.

See Also:

- "XStream Security Model"
- *Oracle Database PL/SQL Packages and Types Reference*

The XStream In Client Application and Security

XStream In allows an application to send LCRs to an inbound server, and an inbound server can apply the database changes in the LCRs to the database.

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` method in the `XStreamIn` class. The connected user is validated as an XStream user.

See Also:

- *Oracle Call Interface Developer's Guide* for information about the OCI interface for XStream
- *Oracle Database XStream Java API Reference* for information about the Java interface for XStream

XStream In Component-Level Security

All the components of the XStream In configuration run as the same user. This user is the apply user for the inbound server.

The `XSTREAM_APPLY` role has privileges required to run components in an XStream In configuration. This role does not contain privileges on the database objects owned by users. If such privileges are required, then they must be granted separately.

See Also:

"Configure an XStream Administrator" for detailed information about configuring an XStream administrator

Privileges Required by the Apply User for an Inbound Server

An inbound server applies LCRs in the security domain of its apply user.

The inbound server receives LCRs from an XStream client application and applies the LCRs that satisfy the inbound server's rule sets. The apply user can apply LCRs directly to database objects. In addition, the apply user runs all custom rule-based transformations specified by the rules in these rule sets. The apply user also runs user-defined apply handlers. XStream In does not assume that the apply user for the inbound server is trusted.

The apply user must have the necessary privileges to apply changes, including the following privileges:

- The required privileges to apply data manipulation language (DML) changes to tables in other schemas (when the inbound server receives DML changes to tables in other schemas)
- The required privileges to apply data definition language (DDL) changes to the database (when the inbound server receives DDL changes)
- `EXECUTE` privilege on the rule sets used by the inbound server
- `EXECUTE` privilege on all custom rule-based transformation functions specified for rules in the positive rule set
- `EXECUTE` privilege on any apply handlers

An inbound server can be associated with only one user, but one user can be associated with many inbound servers.

Grant privileges to the apply user with the `XSTREAM_APPLY` role.

See Also:

- "Configure an XStream Administrator"
- "Changing the Apply User for an Inbound Server"

XStream In and Other Oracle Database Components

XStream In can work with other Oracle Database components.

- XStream In and Oracle Real Application Clusters
  You can configure an inbound server to apply changes in an Oracle Real Application Clusters (Oracle RAC) environment.
- XStream In and Flashback Data Archive
  Inbound servers can apply changes encapsulated in logical change records (LCRs) to tables in a flashback data archive.
- XStream In and Transportable Tablespaces
  You can import data into databases involved in an XStream replication environment using transportable tablespaces.
- XStream In and a Multitenant Environment
  A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.

XStream In and Oracle Real Application Clusters

You can configure an inbound server to apply changes in an Oracle Real Application Clusters (Oracle RAC) environment.

The inbound server runs in the Oracle RAC instance where you connected. In the event that this instance fails, you can connect to a surviving instance and start the inbound server again.

See Also:

- The Inbound Server
- *Oracle Real Application Clusters Administration and Deployment Guide*

XStream In and Flashback Data Archive

Inbound servers can apply changes encapsulated in logical change records (LCRs) to tables in a flashback data archive.

Inbound servers also support the following DDL statements:

- `CREATE FLASHBACK ARCHIVE`
- `ALTER FLASHBACK ARCHIVE`
- `DROP FLASHBACK ARCHIVE`

- `CREATE TABLE` with a `FLASHBACK ARCHIVE` clause
- `ALTER TABLE` with a `FLASHBACK ARCHIVE` clause

See Also:

- The Inbound Server
- *Oracle Database Development Guide* for information about flashback data archive

XStream In and Transportable Tablespaces

You can import data into databases involved in an XStream replication environment using transportable tablespaces.

The instructions in this section apply when the following conditions are met:

- The replication configuration is one in which an inbound server applies changes captured by a capture process in an XStream Out configuration.
- The data being imported with transportable tablespaces must be included in each database in the replication environment.
- After the import operation is complete, changes to the imported data will be replicated.

In addition, the rules should instruct the replication environment to avoid replicating tagged LCRs.

When these conditions are met, complete the following steps:

1. Stop replication.
2. Use transportable tablespaces to import the data into each database in the replication environment.
3. Restart replication.

See Also:

Oracle Database Administrator's Guide for more information about transportable tablespaces

XStream In and a Multitenant Environment

A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.

This self-contained collection is called a pluggable database (PDB). A multitenant container database (CDB) contains PDBs. It can also contain application containers. An application container is an optional component of a CDB that consists of an application root and all application PDBs associated with it. An application container stores data for one or more applications. An application container shares application metadata and common data. In a

CDB, each of the following is a container: the CDB root, each PDB, each application root, and each application PDB.

In a CDB, the inbound server is restricted to receiving LCRs from one source database and only executing changes in the current container (one PDB, one application root, one application PDB, or the CDB root). A single inbound server cannot apply changes to more than one container in a CDB.

When the inbound server is in the CDB root, the apply user must be a common user. When the inbound server is in an application root, the apply user must be a common user or an application common user. When the inbound server is in a PDB or application PDB, the apply user can be a common user or a local user.

Note:

XStream does not synchronize changes done in the application root container. Do not use the XStream In replication to replicate operations done in the application root container. You can manually apply these changes in the application root containers in the target. Note that the operations done in the PDBs can still be replicated.

Related Topics

- System-Created Rules and a Multitenant Environment
  A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.
- *Oracle Multitenant Administrator's Guide*

9

Configuring XStream In

You can configure the Oracle Database components that are used by XStream.

- Preparing for XStream In
  Prerequisites must be met before configuring XStream In.
- Configuring XStream In
  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.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- *Oracle Call Interface Developer's Guide*
- *Oracle Database XStream Java API Reference*

Preparing for XStream In

Prerequisites must be met before configuring XStream In.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- Configure an XStream Administrator
  An XStream administrator configures and manages XStream components in an XStream In environment.
- Set the Relevant Initialization Parameters
  Some initialization parameters are important for the configuration, operation, reliability, and performance of XStream inbound servers. Set these parameters appropriately.
- Configure the Streams pool
  The Streams pool is a portion of memory in the System Global Area (SGA) that is used by both Oracle Streams and XStream components. The Streams pool stores buffered queue LCRs in memory, and it provides memory for inbound servers.

- If Required, Specify Supplemental Logging at the Source Database
  In an XStream configuration in which an inbound server applies changes captured by a capture process in an XStream Out configuration, supplemental logging might be required at the source database on columns in the tables for which an inbound server applies changes.

Configure an XStream Administrator

An XStream administrator configures and manages XStream components in an XStream In environment.

You can configure 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.

If you are configuring XStream In in a multitenant container database (CDB), then configure the XStream administrator in the container that will run the inbound server. This container can be the CDB root, a pluggable database (PDB), an application root, or an application PDB. See "XStream In and a Multitenant Environment" for information about using XStream In in a CDB.

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 user name of the XStream administrator is `xstrmadmin` for a non-CDB. In a CDB, when the XStream administrator is a common user, the user name of the XStream administrator is `c##xstrmadmin`. When the XStream administrator in a CDB is a local user in a container, the user name 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.

   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;
   ```

   If you are creating an XStream administrator as a common user in a CDB, then you must create the tablespace in the CDB root and in all containers. The tablespace is required in all containers because a common user must have access to the tablespace in any container.

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;
   ```

   If you are creating an XStream administrator in a CDB and the inbound server is in the CDB root, then the XStream administrator must be a common user.

   If you are creating an XStream administrator in a CDB and the inbound server is in a PDB, application root, or application PDB, then the XStream administrator can be a common user or a local user. Oracle recommends configuring a common user as the XStream administrator even when the inbound server is in a container other than the CDB root.

   To create a common user, include the `CONTAINER=ALL` clause in the `CREATE USER` statement when the current container is the CDB root:

   ```
   CREATE USER c##xstrmadmin IDENTIFIED BY password
     DEFAULT TABLESPACE xstream_tbs
     QUOTA UNLIMITED ON xstream_tbs
     CONTAINER=ALL;
   ```

   Note:

   Enter an appropriate password for the administrative user.

   See Also:

   Oracle Database Security Guide for guidelines about choosing passwords

4. Grant `CREATE SESSION` privilege to the XStream administrator.

   If you created a new user to act as the XStream administrator, then grant this user `CREATE SESSION` privilege.

   For example, to grant `CREATE SESSION` privilege to user `xstrmadmin`, run the following statement:

   ```
   GRANT CREATE SESSION TO xstrmadmin;
   ```

If you are creating an XStream administrator as a common user in a CDB, then grant `CREATE SESSION` privilege and `SET CONTAINER` privilege to the XStream administrator, and include the `CONTAINER=ALL` clause in the statement.

For example, to grant these privileges to user `xstrmadmin` in a CDB, run the following statement:

```
GRANT CREATE SESSION, SET CONTAINER TO c##xstrmadmin CONTAINER=ALL;
```

5. Grant the `XSTREAM_APPLY` role to the XStream administrator.

   If you are creating an XStream administrator as a common user in a PDB, then run the following statement:

   ```
   GRANT XSTREAM_APPLY to xstrmadmin;
   ```

   If you are creating an XStream administrator in a CDB, run the following statement:

   ```
   GRANT XSTREAM_APPLY to c##xstrmadmin CONTAINER=ALL;
   ```

   See Also:

   Oracle Database PL/SQL Packages and Types Reference

6. If necessary, grant additional privileges to the XStream administrator.

   See "Granting Additional Privileges to the XStream Administrator".

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

Example 9-1 Granting Privileges to a XStream Administrator in PDB

```
GRANT XSTREAM_APPLY to xsadmin;
```

Example 9-2 Granting Privileges to a Trusted XStream Administrator in CDB

In this example, the XStream administrator is a common user.

```
GRANT XSTREAM_APPLY to c##xstrmadmin CONTAINER=ALL;
```

- Granting Additional Privileges to the XStream Administrator
  Additional privileges might be required for the XStream administrator.

Granting Additional Privileges to the XStream Administrator

Additional privileges might be required for the XStream administrator.

Grant any of the following additional privileges to the XStream Administrator if necessary:

- If you plan to use Oracle Enterprise Manager Cloud Control to manage databases with XStream components, then the XStream administrator must be trusted and must be granted `DBA` role. You must also configure the XStream administrator to be an Oracle Enterprise Manager administrative user. Doing so grants additional privileges required by Oracle Enterprise Manager Cloud Control, such as the privileges required to run Oracle Enterprise Manager Cloud Control jobs. See the Oracle Enterprise Manager Cloud Control online help for information about creating Oracle Enterprise Manager administrative users.

- 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 an inbound server. 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.
- If the XStream administrator does not own the queue used by an 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 LCRs into or dequeue LCRs from the queue. The XStream administrator might also need `ENQUEUE` or `DEQUEUE` privileges on the queue, or both.
- 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 you are using Oracle Database Vault, then the following additional privileges are required:
  - 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 `DVSYS.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
      DVSYS.DBMS_MACADM.ADD_AUTH_TO_REALM(
        realm_name  => 'sales',
        grantee     => 'xstrmadmin');
    END;
    /
    ```
  - The user who creates or alters an inbound server must be granted the `BECOME USER` system privilege.

    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 Also:

Oracle Database Vault Administrator's Guide

Set the Relevant Initialization Parameters

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

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 Streams pool

The Streams pool is a portion of memory in the System Global Area (SGA) that is used by both Oracle Streams and XStream components. The Streams pool stores buffered queue LCRs in memory, and it provides memory for inbound servers.

The following are considerations for configuring the Streams pool:

- At least 300 MB of memory is required for the Streams pool.
- The best practice is to set the `STREAMS_POOL_SIZE` initialization parameter explicitly to the desired Streams pool size.
- After XStream In is configured, you can use the `max_sga_size` apply parameter to control the amount of SGA memory allocated specifically to an inbound server.
- Ensure that there is enough space in the Streams pool at each database to run XStream components and to store LCRs and run the components properly.
- The Streams pool is initialized the first time an inbound server is started.

The Streams pool size is the value specified by the `STREAMS_POOL_SIZE` parameter, in bytes, if the following conditions are met:

- The `MEMORY_TARGET`, `MEMORY_MAX_TARGET`, and `SGA_TARGET` initialization parameters are all set to `0` (zero).
- The `STREAMS_POOL_SIZE` initialization parameter is set to a nonzero value.

The Automatic Shared Memory Management feature automatically manages the size of the Streams pool when the following conditions are met:

- The `MEMORY_TARGET` and `MEMORY_MAX_TARGET` initialization parameters are both set to `0` (zero).
- The `SGA_TARGET` initialization parameter is set to a nonzero value.

If you are using Automatic Shared Memory Management, and if the `STREAMS_POOL_SIZE` initialization parameter also is set to a nonzero value, then Automatic Shared Memory Management uses this value as a minimum for the Oracle Streams pool. If your environment needs a minimum amount of memory in the Oracle Streams pool to function properly, then you can set a minimum size. To view the current memory allocated to Oracle Streams pool by Automatic Shared Memory Management, query the `V$SGA_DYNAMIC_COMPONENTS` view. In addition, you can query the `V$STREAMS_POOL_STATISTICS` view to view the current usage of the Oracle Streams pool.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `max_sga_size` apply parameter
- *Oracle Database Administrator's Guide*
- *Oracle Database Reference*

If Required, Specify Supplemental Logging at the Source Database

In an XStream configuration in which an inbound server applies changes captured by a capture process in an XStream Out configuration, supplemental logging might be required at the source database on columns in the tables for which an inbound server applies changes.

The required supplemental logging depends on the configuration of the inbound server you create.

See Also:

"If Required, Configure Supplemental Logging"

Configuring XStream In

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.

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.

Prerequisites

Before configuring XStream In, ensure that the following prerequisite is met:

- Complete the tasks described in "Preparing for XStream In".

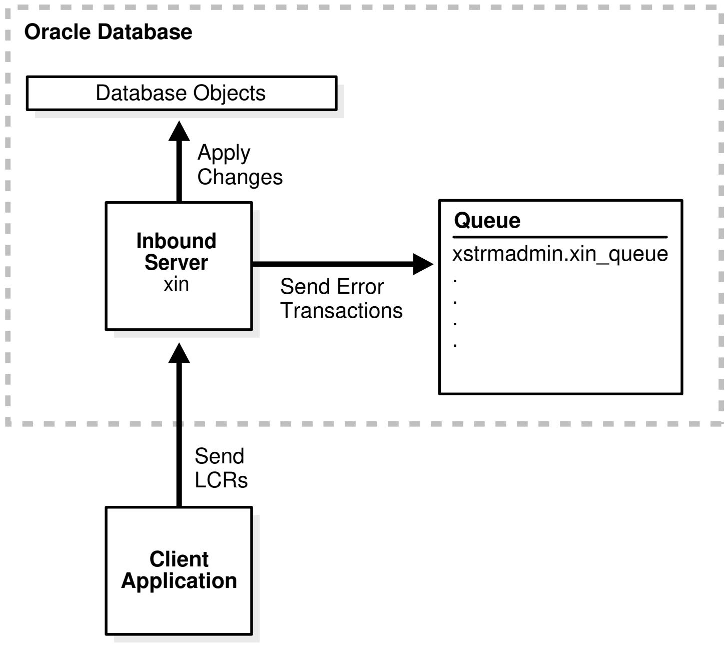
Assumptions for the Sample XStream In Configuration

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 9-1 provides an overview of this XStream In configuration.

Figure 9-1 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.

   If you are configuring XStream In in a CDB, then connect to the container to which the inbound server will apply changes. The container can be the CDB root, a PDB, an application root, or an application PDB. An inbound server can apply changes only in its own container.

   See Also:

   - *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus
   - *Oracle Multitenant Administrator's Guide* for information about connecting to a container in a CDB in SQL*Plus
   - "XStream In and a Multitenant Environment" for information about using XStream In in a CDB

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, the queue 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_XSTREAM_ADM` package or the `DBMS_RULE_ADM` package. See *Oracle Database PL/SQL Packages and Types Reference*.

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:

   "LCR Processing Options for Inbound Servers"

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

   See Also:

   "Sample XStream Client Application" 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:

   Oracle Database PL/SQL Packages and Types Reference

10 Managing XStream In

You can manage an XStream In configuration.

This chapter does not cover using rules, rule sets, or rule-based transformations with inbound servers. By default, an inbound server does not use rules or rule sets. Therefore, an inbound server applies all of the logical change records (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_XSTREAM_ADM` and `DBMS_RULE_ADM` packages. You can also specify rule-based transformations using the `DBMS_XSTREAM_ADM` package.

- About Managing XStream In
  You can modify the database components that are part of an XStream In configuration, such as inbound servers.
- Starting an Inbound Server
  A inbound server must be enabled for it to receive logical change records (LCRs) from an XStream client application and apply the LCRs. You run the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package to start an existing inbound server.
- Stopping an Inbound Server
  You run the `STOP_APPLY` procedure in the `DBMS_APPLY_ADM` package to stop an existing inbound server. You might stop an inbound server when you are troubleshooting a problem in an XStream configuration.
- Setting an Apply Parameter for an Inbound Server
  Apply parameters control the way an inbound server operates. You set an apply parameter for an inbound server using the `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package.
- Changing the Apply User for an Inbound Server
  An inbound server applies LCRs in the security domain of its apply user, and the client application must attach to the inbound server as the apply user. You can change the apply user for an inbound server with the `ALTER_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package.
- Managing XStream In Conflict Detection and Resolution
  When more than one client modifies the same table row at approximately the same time, conflicts are possible. XStream In detects conflicts and provides methods for resolving conflicts.
- Managing Apply Errors
  Apply errors result when an inbound server tries to apply an LCR, and an error is raised.
- Conflict and Error Handling Precedence
  To resolve a conflict or error, an inbound server tries to find conflict handlers and error handlers.
- Dropping Components in an XStream In Configuration
  You can drop an inbound server with the `DROP_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package.

About Managing XStream In

You can modify the database components that are part of an XStream In configuration, such as inbound servers.

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

- `DBMS_XSTREAM_ADM`

  The `DBMS_XSTREAM_ADM` package is the main package for managing XStream In. This package includes subprograms that enable you to configure, modify, or drop inbound servers. This package also enables you modify the rules, rule sets, and rule-based transformations used by inbound servers.

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

- `XSTREAM_APPLY`

  The `XSTREAM_APPLY` role enables you to configure and modify XStream administrators.

  Starting with Oracle Database 23ai, you can manage XStream administrators with the `XSTREAM_APPLY` role. You can start the `REVOKE_ADMIN_PRIVILEGE` procedure by canceling privileges for a user that received XStream privileges prior to the Oracle Database 23ai upgrade.

  > **See Also:**
  >
  > – Configure an XStream Administrator on All Databases for information on creating an XStream administrator
  >
  > – *Oracle Database PL/SQL Packages and Types Reference* for detailed information about this package

- `DBMS_APPLY_ADM`

  The `DBMS_APPLY_ADM` package enables you modify inbound servers.

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

Starting an Inbound Server

A inbound server must be enabled for it to receive logical change records (LCRs) from an XStream client application and apply the LCRs. You run the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package to start an existing inbound server.

To start 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 `START_APPLY` procedure in the `DBMS_APPLY_ADM` package, and specify the inbound server for the `apply_name` parameter.

The following example starts an inbound server named `xin`.

Example 10-1 Starting an Outbound Server Named xout

```
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'xin');
END;
/
```

See Also:

The Oracle Enterprise Manager Cloud Control online help for instructions about starting an apply process or an inbound server with Oracle Enterprise Manager Cloud Control

Stopping an Inbound Server

You run the `STOP_APPLY` procedure in the `DBMS_APPLY_ADM` package to stop an existing inbound server. You might stop an inbound server when you are troubleshooting a problem in an XStream configuration.

To stop 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 `STOP_APPLY` procedure in the `DBMS_APPLY_ADM` package, and specify the inbound server for the `apply_name` parameter.

The following example stops an inbound server named `xin`.

Example 10-2 Stopping an Inbound Server Named xout

```
BEGIN
  DBMS_APPLY_ADM.STOP_APPLY(
    apply_name => 'xin');
```

```
END;
/
```

See Also:

The Oracle Enterprise Manager Cloud Control online help for instructions about stopping an apply process or an inbound server with Oracle Enterprise Manager Cloud Control

Setting an Apply Parameter for an Inbound Server

Apply parameters control the way an inbound server operates. You set an apply parameter for an inbound server using the `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package.

To set an inbound server apply parameter:

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 `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package, and specify the following parameters:
   - `streams_name` - Specify the name of the inbound server.
   - `streams_type` - Specify `apply`.
   - `parameter` - Specify the name of the apply parameter.
   - `value` - Specify the value for the apply parameter.

The following example sets the `parallelism` parameter for an inbound server named `xin` to `4`.

Example 10-3   Setting an Outbound Server Parameter

```
BEGIN
  DBMS_XSTREAM_ADM.SET_PARAMETER(
    streams_name => 'xin',
    streams_type => 'apply',
    parameter    => 'parallelism',
    value        => '4');
END;
/
```

Note:

- The `value` parameter is always entered as a `VARCHAR2` value, even if the parameter value is a number.
- If the `value` parameter is set to `NULL` or is not specified, then the parameter is set to its default value.

See Also:

- The Oracle Enterprise Manager Cloud Control online help for instructions about setting an apply parameter with Oracle Enterprise Manager Cloud Control
- *Oracle Database PL/SQL Packages and Types Reference* for information about apply parameters

Changing the Apply User for an Inbound Server

An inbound server applies LCRs in the security domain of its apply user, and the client application must attach to the inbound server as the apply user. You can change the apply user for an inbound server with 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:

Oracle Database PL/SQL Packages and Types Reference 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 in the `DBMS_XSTREAM_ADM` package, and specify the following parameters:
   - `server_name` - Specify the name of the inbound server.
   - `apply_user` - Specify the new apply user.

Example 10-4 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:

- "XStream In and Security"
- *Oracle Database PL/SQL Packages and Types Reference*

Managing XStream In Conflict Detection and Resolution

When more than one client modifies the same table row at approximately the same time, conflicts are possible. XStream In detects conflicts and provides methods for resolving conflicts.

- About DML Conflicts in an XStream Environment
  A **conflict** is a mismatch between the old values in an LCR and the data in a table.
- Conflict Types in an XStream Environment
  You can encounter several different types of conflicts when you share data at multiple databases.
- Conflicts and Transaction Ordering in an XStream Environment
  Ordering conflicts can occur in an XStream environment when three or more databases share data and the data is updated at two or more of these databases.
- Conflict Detection in an XStream Environment
  An inbound server detects conflicts automatically.
- Conflict Avoidance in an XStream Environment
  There are several ways to avoid data conflicts.
- Conflict Resolution in an XStream Environment
  After an update conflict has been detected, a conflict handler can attempt to resolve it.
- Managing DML Conflict Handlers
  You can set and remove a DML conflict handler. To modify an existing DML conflict handler, you must remove it and reset it.
- Stopping Conflict Detection for Non-Key Columns
  You can stop conflict detection for non-key columns by using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package.

About DML Conflicts in an XStream Environment

A **conflict** is a mismatch between the old values in an LCR and the data in a table.

Conflicts can occur in an XStream environment that permits concurrent data manipulation language (DML) operations on the same data at multiple databases. In an XStream environment, DML conflicts can occur only when an inbound server is applying a row LCR that contains a row change resulting from a DML operation. An inbound server automatically detects conflicts caused by row LCRs.

For example, when two transactions originating at different databases update the same row at nearly the same time, a conflict can occur. When you configure an XStream environment, you must consider whether conflicts can occur. You can configure conflict resolution to resolve conflicts automatically, if your system design permits conflicts.

In general, it is best practice to design an XStream environment that avoids the possibility of conflicts. Using the conflict avoidance techniques discussed later in this chapter, most system designs can avoid conflicts in all or a large percentage of the shared data. However, many applications require that some percentage of the shared data be updatable at multiple databases at any time. If this is the case, then you must address the possibility of conflicts.

Note:

An inbound server does not detect DDL conflicts. Ensure that your environment avoids these types of conflicts.

Related Topics

- Row LCRs
  A row LCR describes a change to the data in a single row or a change to a single LOB column, `LONG` column, `LONG RAW` column, or `XMLType` column in a row.

Conflict Types in an XStream Environment

You can encounter several different types of conflicts when you share data at multiple databases.

- Update Conflicts in an XStream Environment
  An **update conflict** occurs when an inbound server applies a row LCR containing an update to a row that conflicts with another update to the same row.
- Uniqueness Conflicts in an XStream Environment
  A **uniqueness conflict** occurs when an inbound server applies a row LCR containing a change to a row that violates a uniqueness integrity constraint, such as a `PRIMARY KEY` or `UNIQUE` constraint.
- Delete Conflicts in an XStream Environment
  A **delete conflict** occurs when two transactions originate at different databases, with one transaction deleting a row and another transaction updating or deleting the same row.
- Foreign Key Conflicts in an XStream Environment
  A **foreign key conflict** occurs when an inbound server applies a row LCR containing a change to a row that violates a foreign key constraint.

Update Conflicts in an XStream Environment

An **update conflict** occurs when an inbound server applies a row LCR containing an update to a row that conflicts with another update to the same row.

Update conflicts can happen when two transactions originating from different databases update the same row at nearly the same time.

Uniqueness Conflicts in an XStream Environment

A **uniqueness conflict** occurs when an inbound server applies a row LCR containing a change to a row that violates a uniqueness integrity constraint, such as a `PRIMARY KEY` or `UNIQUE` constraint.

For example, consider what happens when two transactions originate from two different databases, each inserting a row into a table with the same primary key value. In this case, the transactions cause a uniqueness conflict.

Delete Conflicts in an XStream Environment

A **delete conflict** occurs when two transactions originate at different databases, with one transaction deleting a row and another transaction updating or deleting the same row.

In this case, the row referenced in the row LCR does not exist and therefore cannot be updated or deleted.

Foreign Key Conflicts in an XStream Environment

A **foreign key conflict** occurs when an inbound server applies a row LCR containing a change to a row that violates a foreign key constraint.

For example, in the `hr` schema, the `department_id` column in the `employees` table is a foreign key of the `department_id` column in the `departments` table. Consider what can happen when the following changes originate at two different databases (`A` and `B`) and are propagated to a third database (`C`):

- At database `A`, a row is inserted into the `departments` table with a `department_id` of `271`. This change is propagated to database `B` and applied there.
- At database `B`, a row is inserted into the `employees` table with an `employee_id` of `206` and a `department_id` of `271`.

If the change that originated at database `B` is applied at database `C` before the change that originated at database `A`, then a foreign key conflict results because the row for the department with a `department_id` of `271` does not yet exist in the `departments` table at database `C`.

Conflicts and Transaction Ordering in an XStream Environment

Ordering conflicts can occur in an XStream environment when three or more databases share data and the data is updated at two or more of these databases.

For example, consider a scenario in which three databases share information in the `hr.departments` table. The database names are `mult1.example.com`, `mult2.example.com`, and `mult3.example.com`. Suppose a change is made to a row in the `hr.departments` table at `mult1.example.com` that will be propagated to both `mult2.example.com` and `mult3.example.com`. The following series of actions might occur:

1. The change is propagated to `mult2.example.com`.
2. An inbound server at `mult2.example.com` applies the change from `mult1.example.com`.
3. A different change to the same row is made at `mult2.example.com`.
4. The change at `mult2.example.com` is propagated to `mult3.example.com`.
5. An inbound server at `mult3.example.com` attempts to apply the change from `mult2.example.com` before another inbound server at `mult3.example.com` applies the change from `mult1.example.com`.

In this case, a conflict occurs because a column value for the row at `mult3.example.com` does not match the corresponding old value in the row LCR propagated from `mult2.example.com`.

In addition to causing a data conflict, transactions that are applied out of order might experience referential integrity problems at a remote database if supporting data has not been successfully propagated to that database. Consider the scenario where a new customer calls an order department. A customer record is created and an order is placed. If the order data is applied at a remote database before the customer data, then a referential integrity error is raised because the customer that the order references does not exist at the remote database.

If an ordering conflict is encountered, then you can resolve the conflict by reexecuting the transaction in the error queue after the required data has been propagated to the remote database and applied.

Conflict Detection in an XStream Environment

An inbound server detects conflicts automatically.

- About Conflict Detection in an XStream Environment
  An inbound server detects update, uniqueness, delete, and foreign key conflicts.
- Control Over Conflict Detection for Non-Key Columns
  By default, an inbound server compares old values for all columns during conflict detection, but you can stop conflict detection for non-key columns using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package.
- Rows Identification During Conflict Detection in an XStream Environment
  To detect conflicts accurately, Oracle Database must be able to identify and match corresponding rows at different databases uniquely.

About Conflict Detection in an XStream Environment

An inbound server detects update, uniqueness, delete, and foreign key conflicts.

An inbound server detects these conflicts as follows:

- An inbound server detects an update conflict if there is any difference between the old values for a row in a row LCR and the current values of the same row at the destination database.
- An inbound server detects a uniqueness conflict if a uniqueness constraint violation occurs when applying an LCR that contains an insert or update operation.
- An inbound server detects a delete conflict if it cannot find a row when applying an LCR that contains an update or delete operation, because the primary key of the row does not exist.
- An inbound server detects a foreign key conflict if a foreign key constraint violation occurs when applying an LCR.

A conflict can be detected when an inbound server attempts to apply an LCR directly or when an inbound server handler, such as a DML conflict handler, runs the `EXECUTE` member procedure for an LCR. A conflict can also be detected when either the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package is run.

Note:

- If a column is updated and the column's old value equals its new value, then Oracle Database never detects a conflict for this column update.
- Any old LOB values in update LCRs, delete LCRs, and LCRs dealing with piecewise updates to LOB columns are not used by conflict detection.

Control Over Conflict Detection for Non-Key Columns

By default, an inbound server compares old values for all columns during conflict detection, but you can stop conflict detection for non-key columns using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package.

Conflict detection might not be needed for some non-key columns.

See Also:

"Stopping Conflict Detection for Non-Key Columns"

Rows Identification During Conflict Detection in an XStream Environment

To detect conflicts accurately, Oracle Database must be able to identify and match corresponding rows at different databases uniquely.

By default, Oracle Database uses the primary key of a table to identify rows in a table uniquely. When a table does not have a primary key, it is best practice to designate a substitute key. A substitute key is a column or set of columns that Oracle Database can use to identify uniquely rows in the table.

Conflict Avoidance in an XStream Environment

There are several ways to avoid data conflicts.

- Use a Primary Database Ownership Model
  You can avoid the possibility of conflicts by limiting the number of databases that have simultaneous update access to the tables containing shared data.
- Avoid Specific Types of Conflicts
  If a primary database ownership model is too restrictive for your application requirements, then you can use a shared ownership data model, which means that conflicts might be possible. Even so, typically you can use some simple strategies to avoid specific types of conflicts.

Use a Primary Database Ownership Model

You can avoid the possibility of conflicts by limiting the number of databases that have simultaneous update access to the tables containing shared data.

Primary ownership prevents all conflicts, because only a single database permits updates to a set of shared data. Applications can even use row and column subsetting to establish more granular ownership of data than at the table level. For example, applications might have update access to specific columns or rows in a shared table on a database-by-database basis.

Avoid Specific Types of Conflicts

If a primary database ownership model is too restrictive for your application requirements, then you can use a shared ownership data model, which means that conflicts might be possible. Even so, typically you can use some simple strategies to avoid specific types of conflicts.

- Avoid Uniqueness Conflicts in an XStream Environment
  You can avoid uniqueness conflicts by ensuring that each database uses unique identifiers for shared data.
- Avoid Delete Conflicts in an Oracle Replication Environment
  Always avoid delete conflicts in shared data environments.
- Avoid Update Conflicts in an XStream Environment
  After trying to eliminate the possibility of uniqueness and delete conflicts, you should also try to limit the number of possible update conflicts.

Avoid Uniqueness Conflicts in an XStream Environment

You can avoid uniqueness conflicts by ensuring that each database uses unique identifiers for shared data.

There are three ways to ensure unique identifiers at all databases in an XStream environment.

- One way is to construct a unique identifier by executing the following select statement:

  ```
  SELECT SYS_GUID() OID FROM DUAL;
  ```

  This SQL operator returns a 16-byte globally unique identifier. The globally unique identifier appears in a format similar to the following:

  ```
  A741C791252B3EA0E034080020AE3E0A
  ```

- Another way to avoid uniqueness conflicts is to create a sequence at each of the databases that shares data and concatenate the database name (or other globally unique value) with the local sequence. This approach helps to avoid any duplicate sequence values and helps to prevent uniqueness conflicts.

- Finally, you can create a customized sequence at each of the databases that shares data so that no two databases can generate the same value. You can accomplish this by using a combination of starting, incrementing, and maximum values in the `CREATE SEQUENCE` statement. For example, you might configure the following sequences:

  Table 10-1 Customized Sequences

Parameter	Database A	Database B	Database C
`START WITH`	1	3	5
`INCREMENT BY`	10	10	10
Range Example	1, 11, 21, 31, 41,...	3, 13, 23, 33, 43,...	5, 15, 25, 35, 45,...

  Using a similar approach, you can define different ranges for each database by specifying a `START WITH` and `MAXVALUE` that would produce a unique range for each database.

Avoid Delete Conflicts in an Oracle Replication Environment

Always avoid delete conflicts in shared data environments.

In general, it is best practice for applications that operate within a shared ownership data model to avoid deleting rows using `DELETE` statements. Instead, applications can mark rows for deletion and then configure the system to purge logically deleted rows periodically.

Avoid Update Conflicts in an XStream Environment

After trying to eliminate the possibility of uniqueness and delete conflicts, you should also try to limit the number of possible update conflicts.

However, in a shared ownership data model, update conflicts cannot be avoided in all cases. If you cannot avoid all update conflicts, then you must understand the types of conflicts possible and configure the system to resolve them if they occur.

Conflict Resolution in an XStream Environment

After an update conflict has been detected, a conflict handler can attempt to resolve it.

- About Conflict Resolution in an XStream Environment
  XStream provides prebuilt conflict handlers to resolve insert and update conflicts.
- Prebuilt DML Conflict Handlers
  There are several types of prebuilt DML conflict handlers available. Column lists and resolution columns are used in prebuilt DML conflict handlers.
- Types of Prebuilt DML Conflict Handlers
  Oracle provides the following types of prebuilt DML conflict handlers for an Oracle Replication environment: `RECORD`, `IGNORE`, `OVERWRITE`, `MAXIMUM`, `MINIMUM`, and `DELTA`.
- Column Lists
  Each time you specify a prebuilt DML conflict handler for a table, you must specify a column list.
- Resolution Columns
  The **resolution column** is the column used to identify a prebuilt DML conflict handler.
- Data Convergence
  When you share data between multiple databases, and you want the data to be the same at all of these databases, ensure that you use conflict resolution handlers that cause the data to converge at all databases.
- Collision Handling Without a DML Conflict Handler
  In the absence of a DML conflict handler for a table, you can enable basic collision handling using the `HANDLE_COLLISIONS` procedure in the `DBMS_APPLY_ADM` package.
- Custom Conflict Handlers
  You can create a PL/SQL procedure to use as a custom conflict handler.

About Conflict Resolution in an XStream Environment

XStream provides prebuilt conflict handlers to resolve insert and update conflicts.

There are no prebuilt conflict handlers for delete, foreign key, or ordering conflicts. However, you can build your own custom conflict handler to resolve data conflicts specific to your business rules. Such a conflict handler can be part of a procedure DML handler or an error handler.

Whether you use prebuilt or custom conflict handlers, a conflict handler is applied as soon as a conflict is detected. If neither the specified conflict handler nor the relevant apply handler can resolve the conflict, then the conflict is logged in the error queue. You might want to use the relevant apply handler to notify the database administrator when a conflict occurs.

When a conflict causes a transaction to be moved to the error queue, sometimes it is possible to correct the condition that caused the conflict. In these cases, you can reexecute a transaction using the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package.

See Also:

*Oracle Database PL/SQL Packages and Types Reference*for more information about the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package

Prebuilt DML Conflict Handlers

There are several types of prebuilt DML conflict handlers available. Column lists and resolution columns are used in prebuilt DML conflict handlers.

A column list is a list of columns for which the DML conflict handler is called when there is an insert or update conflict. The resolution column identifies a DML conflict handler. If you use a `MAXIMUM` or `MINIMUM` prebuilt DML conflict handler, then the resolution column is also the column used to resolve the conflict. The resolution column must be one of the columns in the column list for the handler.

Use the `SET_DML_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package to specify one or more DML conflict handlers for a particular table. There are no prebuilt DML conflict handlers for delete or foreign key conflicts.

See Also:

- "Managing DML Conflict Handlers" for instructions on setting and removing an DML conflict handler
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `SET_DML_CONFLICT_HANDLER` procedure
- "Column Lists"
- "Resolution Columns"

Types of Prebuilt DML Conflict Handlers

Oracle provides the following types of prebuilt DML conflict handlers for an Oracle Replication environment: `RECORD`, `IGNORE`, `OVERWRITE`, `MAXIMUM`, `MINIMUM`, and `DELTA`.

The description for each type of handler later in this topic refers to the following conflict scenario:

1. The following update is made at the `dbs1.example.com` source database:

   ```
   UPDATE hr.employees SET salary = 4900 WHERE employee_id = 200;
   COMMIT;
   ```

   This update changes the salary for employee `200` from `4400` to `4900`.

2. At nearly the same time, the following update is made at the `dbs2.example.com` destination database:

   ```
   UPDATE hr.employees SET salary = 5000 WHERE employee_id = 200;
   COMMIT;
   ```

3. A capture process captures the update at the `dbs1.example.com` source database and puts the resulting row LCR in a queue.
4. A propagation propagates the row LCR from the queue at `dbs1.example.com` to a queue at `dbs2.example.com`.
5. An apply process at `dbs2.example.com` attempts to apply the row LCR to the `hr.employees` table but encounters a conflict because the salary value at `dbs2.example.com` is `5000`, which does not match the old value for the salary in the row LCR (`4400`).

The following sections describe each prebuilt conflict handler and explain how the handler resolves this conflict.

RECORD

When a conflict occurs, the `RECORD` handler places the LCR into the error queue. The `RECORD` handler can be used for all conflict types, but it can only be specified for a column group that contains all the columns in the table.

If the `RECORD` handler is used for the `hr.employees` table at the `dbs2.example.com` destination database in the conflict example, then the row LCR from `dbs1.example.com` is placed in the error queue at `dbs1.example.com`, and its changes are not applied. Therefore, after the conflict is resolved, the salary for employee `200` is `5000` at `dbs2.example.com`.

IGNORE

When a conflict occurs, the `IGNORE` handler ignores the values in the LCR from the source database and retains the value at the destination database.

If the `IGNORE` handler is used for the `hr.employees` table at the `dbs2.example.com` destination database in the conflict example, then the new value in the row LCR is discarded. Therefore, after the conflict is resolved, the salary for employee `200` is `5000` at `dbs2.example.com`.

OVERWRITE

When a conflict occurs, the `OVERWRITE` handler replaces the current value at the destination database with the new value in the LCR from the source database.

If the `OVERWRITE` handler is used for the `hr.employees` table at the `dbs2.example.com` destination database in the conflict example, then the new value in the row LCR overwrites the value at `dbs2.example.com`. Therefore, after the conflict is resolved, the salary for employee `200` is `4900`.

MAXIMUM

When a conflict occurs, the `MAXIMUM` conflict handler compares the new value in the LCR from the source database with the current value in the destination database for a designated resolution column. If the new value of the resolution column in the LCR is greater than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the LCR. If the new value of the resolution column in the LCR is less than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the destination database.

If the MAXIMUM handler is used for the salary column in the hr.employees table at the dbs2.example.com destination database in the conflict example, then the apply process does not apply the row LCR, because the salary in the row LCR is less than the current salary in the table. Therefore, after the conflict is resolved, the salary for employee 200 is 5000 at dbs2.example.com.

If you want to resolve conflicts based on the time of the transactions involved, then one way to do this is to add a column to a shared table that automatically records the transaction time with a trigger. You can designate this column as a resolution column for a MAXIMUM conflict handler, and the transaction with the latest (or greater) time would be used automatically.

The following is an example of a trigger that records the time of a transaction for the hr.employees table. Assume that the job_id, salary, and commission_pct columns are part of the column list for the conflict resolution handler. The trigger should fire only when an UPDATE is performed on the columns in the column list or when an INSERT is performed.

```
ALTER TABLE hr.employees ADD (time TIMESTAMP WITH TIME ZONE);

CREATE OR REPLACE TRIGGER hr.insert_time_employees
BEFORE
  INSERT OR UPDATE OF job_id, salary, commission_pct ON hr.employees
FOR EACH ROW
BEGIN
   -- Consider time synchronization problems. The previous update to this
   -- row might have originated from a site with a clock time ahead of the
   -- local clock time.
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/
```

If you use such a trigger for conflict resolution, then ensure that the trigger's firing property is “fire once,” which is the default. Otherwise, a new time might be marked when transactions are applied by an apply process, resulting in the loss of the actual time of the transaction.

MINIMUM

When a conflict occurs, the MINIMUM conflict handler compares the new value in the LCR from the source database with the current value in the destination database for a designated resolution column. If the new value of the resolution column in the LCR is less than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the LCR. If the new value of the resolution column in the LCR is greater than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the destination database.

If the MINIMUM handler is used for the salary column in the hr.employees table at the dbs2.example.com destination database in the conflict example, then the apply process resolves the conflict in favor of the row LCR, because the salary in the row LCR is less than the current salary in the table. Therefore, after the conflict is resolved, the salary for employee 200 is 4900.

DELTA

When a conflict occurs, the DELTA conflict handler calculates the difference between the old value for the column and the new value for the column and adds the difference to the current value of the column. The `DELTA` conflict handler can only be used when the `conflict_type` is set to `ROW_EXISTS` and all of the columns in the column group are numbers.

If the `DELTA` handler is used for the `salary` column in the `hr.employees` table at the `dbs2.example.com` destination database in the conflict example, then the apply process resolves the conflict by calculating the difference between the old value for the column and the new value for the column (4900 – 4400 = 500) and adding it to the current value of the column (5000 + 500 = 5500). Therefore, after the conflict is resolved, the salary for employee `200` is `5500`.

MAX_AND_EQUALS

When a conflict occurs, apply the column list from in the LCR if the value of resolution column is greater than or equal to the value of the column in the database. Otherwise, discard the LCR.

If the `MAX_AND_EQUALS` handler is used for the `salary` column in the `hr.employees` table at the `dbs2.example.com` destination database in the conflict example, then the apply process resolves the conflict by discarding the LCR. Therefore, after the conflict is resolved, the salary for employee `200` is `5000`.

MIN_AND_EQUALS

When a conflict occurs, apply the column list from the LCR if the value of resolution column is less than or equal to the value of the column in the database. Otherwise, discard the LCR.

If the `MIN_AND_EQUALS` handler is used for the `salary` column in the `hr.employees` table at the `dbs2.example.com` destination database in the conflict example, then the apply process resolves the conflict by applying the LCR. Therefore, after the conflict is resolved, the salary for employee `200` is `4900`.

Column Lists

Each time you specify a prebuilt DML conflict handler for a table, you must specify a column list.

A column list is a list of columns for which the DML conflict handler is called. If an update conflict occurs for one or more of the columns in the list when an inbound server tries to apply a row LCR, then the DML conflict handler is called to resolve the conflict. The DML conflict handler is not called if a conflict occurs only in columns that are not in the list. The scope of conflict resolution is a single column list on a single row LCR.

You can specify multiple DML conflict handlers for a particular table, but the same column cannot be in more than one column list. For example, suppose you specify two prebuilt DML conflict handlers on `hr.employees` table:

- The first DML conflict handler has the following columns in its column list: `salary` and `commission_pct`.
- The second DML conflict handler has the following columns in its column list: `job_id` and `department_id`.

Also, assume that no other conflict handlers exist for this table. In this case, the following examples illustrate the outcomes for different scenarios:

- If a conflict occurs for the `salary` column when an inbound server tries to apply a row LCR, then the first DML conflict handler is called to resolve the conflict.

- If a conflict occurs for the `department_id` column, then the second DML conflict handler is called to resolve the conflict.
- If a conflict occurs for a column that is not in a column list for any conflict handler, then no conflict handler is called, and an error results. For instance, if a conflict occurs for the `manager_id` column in the `hr.employees` table, then an error results.
- If conflicts occur in more than one column list when a row LCR is being applied, and there are no conflicts in any columns that are not in a column list, then the appropriate DML conflict handler is invoked for each column list with a conflict.

Column lists enable you to use different handlers to resolve conflicts for different types of data. For example, numeric data is often suited for a maximum or minimum conflict handler, while an overwrite or discard conflict handler might be preferred for character data.

If a conflict occurs in a column that is not in a column list, then the error handler for the specific operation on the table attempts to resolve the conflict. If the error handler cannot resolve the conflict, or if there is no such error handler, then the transaction that caused the conflict is moved to the error queue.

Also, if a conflict occurs for a column in a column list that uses either the `OVERWRITE`, `MAXIMUM`, or `MINIMUM` prebuilt handler, and if the row LCR does not contain all of the columns in this column list, then the conflict cannot be resolved because all of the values are not available. In this case, the transaction that caused the conflict is moved to the error queue. If the column list uses the `DISCARD` prebuilt method, then the row LCR is discarded and no error results, even if the row LCR does not contain all of the columns in this column list.

If more than one column at the source database affects the column list at the destination database, then a conditional supplemental log group must be specified for the columns specified in a column list. Supplemental logging is specified at the source database and adds additional information to the LCR, which is needed to resolve conflicts properly. Typically, a conditional supplemental log group must be specified for the columns in a column list if there are multiple columns in the column list, but not if there is only one column in the column list.

However, in some cases, a conditional supplemental log group is required even if there is only one column in a column list. That is, an apply handler or custom rule-based transformation can combine multiple columns from the source database into a single column in the column list at the destination database. For example, a custom rule-based transformation can take three columns that store street, state, and postal code data from a source database and combine the data into a single address column at a destination database.

Also, in some cases, no conditional supplemental log group is required even if there are multiple columns in a column list. For example, an apply handler or custom rule-based transformation can separate one address column from the source database into multiple columns that are in a column list at the destination database. A custom rule-based transformation can take an address that includes street, state, and postal code data in one address column at a source database and separate the data into three columns at a destination database.

Note:

Prebuilt DML conflict handlers do not support LOB, `LONG`, `LONG RAW`, user-defined type, and Oracle-supplied type columns. Therefore, you should not include these types of columns in the `column_list` parameter when running the `SET_DML_CONFLICT_HANDLER` procedure.

See Also:

- "If Required, Specify Supplemental Logging at the Source Database"
- *Oracle Database SQL Language Reference* for information about data types

Resolution Columns

The **resolution column** is the column used to identify a prebuilt DML conflict handler.

If you use a `MAXIMUM` or `MINIMUM` prebuilt DML conflict handler, then the resolution column is also the column used to resolve the conflict. The resolution column must be one of the columns in the column list for the handler.

For example, if the `salary` column in the `hr.employees` table is specified as the resolution column for a maximum or minimum conflict handler, then the `salary` column is evaluated to determine whether column list values in the row LCR are applied or the destination database values for the column list are retained.

In either of the following situations involving a resolution column for a conflict, the apply process moves the transaction containing the row LCR that caused the conflict to the error queue, if the error handler cannot resolve the problem. In these cases, the conflict cannot be resolved and the values of the columns at the destination database remain unchanged:

- The new LCR value and the destination row value for the resolution column are the same (for example, if the resolution column was not the column causing the conflict).
- Either the new LCR value of the resolution column or the current value of the resolution column at the destination database is `NULL`.

Note:

Although the resolution column is not used for `OVERWRITE` and `DISCARD` conflict handlers, you must specify a resolution column for these conflict handlers.

Data Convergence

When you share data between multiple databases, and you want the data to be the same at all of these databases, ensure that you use conflict resolution handlers that cause the data to converge at all databases.

If you allow changes to shared data at all of your databases, then data convergence for a table is possible only if all databases that are sharing data capture changes to the shared data and propagate these changes to all of the other databases that are sharing the data.

In such an environment, the `MAXIMUM` conflict resolution method can guarantee convergence only if the values in the resolution column are always increasing. If successive time stamps on a row are distinct, then a time-based resolution column meets this requirement. The `MINIMUM` conflict resolution method can guarantee convergence in such an environment only if the values in the resolution column are always decreasing.

Collision Handling Without a DML Conflict Handler

In the absence of a DML conflict handler for a table, you can enable basic collision handling using the `HANDLE_COLLISIONS` procedure in the `DBMS_APPLY_ADM` package.

When you enable basic collision handling for an inbound server and a table, conflicts are resolved in the following ways:

- When a conflict is detected for a row that exists in the table, the data in the row LCR overwrites the data in the table.

  For example, if a row LCR contains an insert, but the row already exists in the table. The data in the row LCR overwrites the existing data in the table. If a row LCR contains an update, and an old value in the row does not match an old value in the row LCR, the data in the row LCR overwrites the data in the table.
- When a conflict is detected for a row that does not exist in the table, the data in the row LCR is ignored.

  For example, if a row LCR contains an update to a row, but the row does not exist in the table, the row LCR is ignored.

Example 10-5 Enabling Basic Collision Handling for a Table

```
app_emphr.employees

BEGIN
  DBMS_APPLY_ADM.HANDLE_COLLISIONS(
    apply_name => 'app_emp',
    enable     => TRUE,
    object     => 'hr.employees');
END;
/
```

To disable basic collision handling for this table, run the same procedure, but set the `enable` parameter to `FALSE`.

Custom Conflict Handlers

You can create a PL/SQL procedure to use as a custom conflict handler.

You use the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package to designate one or more custom conflict handlers for a particular table. Specifically, set the following parameters when you run this procedure to specify a custom conflict handler:

- Set the `object_name` parameter to the fully qualified name of the table for which you want to perform conflict resolution.
- Set the `object_type` parameter to `TABLE`.
- Set the `operation_name` parameter to the type of operation for which the custom conflict handler is called. The possible operations are the following: `INSERT`, `UPDATE`, `DELETE`, and `LOB_UPDATE`. You can also set the `operation_name` parameter to `DEFAULT` so that the handler is the default handler for all operations.
- If you want an error handler to perform conflict resolution when an error is raised, then set the `error_handler` parameter to `TRUE`. Or, if you want to include conflict resolution in your procedure DML handler, then set the `error_handler` parameter to `FALSE`.

If you specify FALSE for this parameter, then, when you execute a row LCR using the EXECUTE member procedure for the LCR, the conflict resolution within the procedure DML handler is performed for the specified object and operation(s).

- Specify the procedure to resolve a conflict by setting the user_procedure parameter. This user procedure is called to resolve any conflicts on the specified table resulting from the specified type of operation.

If the custom conflict handler cannot resolve the conflict, then the inbound server moves the transaction containing the conflict to the error queue and does not apply the transaction.

If both a prebuilt DML conflict handler and a custom conflict handler exist for a particular object, then the prebuilt DML conflict handler is invoked only if both of the following conditions are met:

- The custom conflict handler executes the row LCR using the EXECUTE member procedure for the LCR.
- The conflict_resolution parameter in the EXECUTE member procedure for the row LCR is set to TRUE.

See Also:

*Oracle Database PL/SQL Packages and Types Reference*for more information about the SET_DML_HANDLER procedure

Managing DML Conflict Handlers

You can set and remove a DML conflict handler. To modify an existing DML conflict handler, you must remove it and reset it.

- Setting a DML Conflict Handler
  Set a DML conflict handler using the SET_DML_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package.
- Removing a DML Conflict Handler
  You can remove an existing DML conflict handler by running the SET_DML_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package.

Setting a DML Conflict Handler

Set a DML conflict handler using the SET_DML_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package.

You can use one of the following prebuilt methods when you create a DML conflict resolution handler:

- RECORD
- IGNORE
- OVERWRITE
- MAXIMUM
- MINIMUM

- `DELTA`
- `MAX_AND_EQUALS`
- `MIN_AND_EQUALS`

To set a DML conflict handler:

1. Connect to the inbound server database as the XStream administrator.
2. Run the `SET_DML_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package.

Example 10-6 Setting DML Conflict Handlers

Suppose an XStream In client receives changes to be applied to the `hr.jobs` table at `dbs1.example.com`. In this environment, conflicts can occur because the changes from the external database that the client receives may not be coordinated with the changes to the target database `dbs1.example.com`. If there is a conflict for a particular DML insert or update, then the change from the external database must always overwrite the change at the target database. In this environment, you can accomplish this goal by specifying an `OVERWRITE` handler at the `dbs1.example.com` database. If there is a conflict because the row for a DML delete does not exist, then the row LCR is ignored.

This example specifies DML conflict handlers for the `hr.jobs` table at the `dbs1.example.com` database.

```
DECLARE
  cols DBMS_UTILITY.LNAME_ARRAY;
  BEGIN
    cols(1) := 'job_title';
    cols(2) := 'min_salary';
    cols(3) := 'max_salary';
    DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
      apply_name            => 'app_jobs',
      conflict_handler_name => 'jobs_handler_insert',
      object                => 'hr.jobs',
      operation_name        => 'INSERT',
      conflict_type         => 'ROW_EXISTS',
      method_name           => 'OVERWRITE',
      column_table          => cols);
    DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
      apply_name            => 'app_jobs',
      conflict_handler_name => 'jobs_handler_update',
      object                => 'hr.jobs',
      operation_name        => 'UPDATE',
      conflict_type         => 'ROW_EXISTS',
      method_name           => 'OVERWRITE',
      column_table          => cols);
    DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
      apply_name            => 'app_jobs',
      conflict_handler_name => 'jobs_handler_delete',
      object                => 'hr.jobs',
      operation_name        => 'DELETE',
      conflict_type         => 'ROW_MISSING',
      method_name           => 'IGNORE',
      column_list           => '*');
END;
/
```

The apply process app_jobs uses the specified DML conflict handlers.

Note:

- For the jobs_handler_delete DML conflict handler, the column_list parameter is set to '*' because all columns must be specified when the operation_name is set to DELETE.
- If the client is obtaining data from an Oracle database using XStream Out, then you must specify a conditional supplemental log group at the source database for all of the columns in the column_list at the destination database. In this example, you would specify a conditional supplemental log group including the job_title, min_salary, and max_salary columns in the hr.jobs table at the external database.
- Prebuilt DML conflict handlers do not support LOB, LONG, LONG RAW, user-defined type, and Oracle-supplied type columns. Therefore, do not include these types of columns in the column_list parameter when running the procedure SET_DML_CONFLICT_HANDLER.

See Also:

Oracle Database SQL Language Reference for information about data types

Removing a DML Conflict Handler

You can remove an existing DML conflict handler by running the SET_DML_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package.

To remove an existing DML conflict handler, specify NULL for the method, and specify the same apply name and DML conflict handler name as the existing DML conflict handler.

To remove a DML conflict handler:

1. Connect to the inbound server database as the XStream administrator.
2. Run the SET_DML_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package with NULL specified for the method, and specify the same apply name, DML conflict handler name, object name, conflict type, and resolution column as the existing DML conflict handler.

Example 10-7 Removing a DML Conflict Handler

To remove the DML conflict handler created in "Setting a DML Conflict Handler", run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
     apply_name            => 'app_jobs',
     conflict_handler_name => 'jobs_handler_insert',
     method_name           => NULL);
  DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
     apply_name            => 'app_jobs',
     conflict_handler_name => 'jobs_handler_update',
     method_name           => NULL);
  DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
     apply_name            => 'app_jobs',
     conflict_handler_name => 'jobs_handler_delete',
     method_name           => NULL);
END;
/
```

Stopping Conflict Detection for Non-Key Columns

You can stop conflict detection for non-key columns by using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package.

To stop conflict detection for non-key columns:

1. Connect to the inbound server database as the XStream administrator.
2. Run the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package, and specify the non-key columns and `FALSE` for the `compare` parameter.

Example 10-8   Stopping Conflict Detection for Non-Key Columns

Suppose you configure a `time` column for conflict resolution for the `hr.employees` table. A trigger records the current time in this column for each change to the table. In this case, you can decide to stop conflict detection for the other non-key columns in the table. Add the columns in the `hr.employees` table to the column list for an update conflict handler:

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1)  := 'first_name';
  cols(2)  := 'last_name';
  cols(3)  := 'email';
  cols(4)  := 'phone_number';
  cols(5)  := 'hire_date';
  cols(6)  := 'job_id';
  cols(7)  := 'salary';
  cols(8)  := 'commission_pct';
  cols(9)  := 'manager_id';
  cols(10) := 'department_id';
  cols(11) := 'time';
  DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
    apply_name            => 'app_employees',
    conflict_handler_name => 'emp_handler',
    object                => 'hr.employees',
    operation_name        => 'UPDATE',
    conflict_type         => 'ROW_EXISTS',
    method_name           => 'MAXIMUM',
    column_list           => cols,
    resolution_column     => 'time');
END;
/
```

This example does not include the primary key for the table in the column list because it assumes that the primary key is never updated. However, other key columns are included in the column list.

To stop conflict detection for all non-key columns in the table for `UPDATE` operations, enter the following:

```
DECLARE
  cols DBMS_UTILITY.LNAME_ARRAY;
  BEGIN
    cols(1) := 'first_name';
    cols(2) := 'last_name';
    cols(3) := 'email';
    cols(4) := 'phone_number';
    cols(5) := 'hire_date';
    cols(6) := 'job_id';
    cols(7) := 'salary';
    cols(8) := 'commission_pct';
  DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
    object_name  => 'hr.employees',
    column_table => cols,
    operation    => '*',
    compare      => FALSE);
END;
/
```

The asterisk (*) specified for the operation parameter means that conflict detection is stopped for `UPDATE` operations. After you run this procedure, all apply processes running on the database that apply changes to the specified table locally do not detect conflicts on the specified columns. Therefore, in this example, the `time` column is the only column used for conflict detection.

Note:

The example in this section sets an DML conflict handler before stopping conflict detection for non-key columns. However, a DML conflict handler is not required before you stop conflict detection for non-key columns.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `COMPARE_OLD_VALUES` procedure

Managing Apply Errors

Apply errors result when an inbound server tries to apply an LCR, and an error is raised.

When an apply error occurs, the LCR that caused the error and all of the other LCRs in the same transaction are moved to the error queue.

- Inbound Server Error Handling
  You can configure error handlers to handle specific types of errors.
- Retrying Apply Error Transactions
  You can retry a specific error transaction, or you can retry all error transactions for an inbound server.
- Deleting Apply Error Transactions
  You can delete a specific error transaction, or you can delete all error transactions for an inbound server.
- Managing Eager Errors Encountered by an Inbound Server
  As a performance optimization, an inbound server can use eager apply to begin to apply large transactions before it receives the commit LCR.

See Also:

- "The Error Queue for an Inbound Server"
- The Oracle Enterprise Manager Cloud Control online help for instructions on managing apply errors in Oracle Enterprise Manager Cloud Control

Inbound Server Error Handling

You can configure error handlers to handle specific types of errors.

- About Error Handlers
  An error handler specifies a method for handling a specific error during apply.
- Setting and Unsetting an Error Handler
  You set an error handler with the `SET_REPERROR_HANDLER` procedure in the `DBMS_APPLY` package.

About Error Handlers

An error handler specifies a method for handling a specific error during apply.

When an inbound server applies row LCRs, it can encounter errors. You can configure an error handler to handle a specific error using a designated method with the `SET_REPERROR_HANDLER` procedure in the `DBMS_APPLY` package. For example, you can set an error handler that handles ORA-26787 errors that occur when a row LCR tries to update or delete a row that does not exist in a table. In addition, you can configure a default error handling method without specifying a particular error.

You set an error handler for a specific apply process. You can set an error handler for a specific table or for all tables.

The following table describes each error handler method.

Table 10-2 Error Handler Methods

Method	Description
`ABEND`	Stop the inbound server when the error is encountered.

Table 10-2 (Cont.) Error Handler Methods

Method	Description
`RECORD`	Move the row LCR that caused the error to the error queue when the error is encountered.
`IGNORE`	Silently ignore the error, and do not apply the row LCR, when the error is encountered.
`RETRY`	Retry the row LCR for the specified number of times when the error is encountered. If retry fails, then the entire transaction is moved to the error queue.
`RETRY_TRANSACTION`	Retry the transaction for the specified number of times, with the specified delay before retry, when the error is encountered. If retry fails, then the entire transaction is moved to the error queue.
`RECORD_TRANSACTION`	Move the entire transaction to the error queue when the error is encountered. `RECORD_TRANSACTION` is the default.

Setting and Unsetting an Error Handler

You set an error handler with the `SET_REPERROR_HANDLER` procedure in the `DBMS_APPLY` package.

You can use one of the following methods when you set an error handler:

- `ABEND`
- `RECORD`
- `IGNORE`
- `RETRY`
- `RETRY_TRANSACTION`
- `RECORD_TRANSACTION`

To unset an error handler, set the `method` parameter in the `SET_REPERROR_HANDLER` procedure to `NULL`.

To set or unset an error handler:

1. Connect to the inbound server database as the XStream administrator.
2. Run the `SET_REPERROR_HANDLER` procedure in the `DBMS_APPLY_ADM` package.

Example 10-9 Setting an Error Handler That Stops the Inbound Server for All Errors on a Specific Table

This example sets an error handler that stops the `app_oe` inbound server for any errors on the `oe.orders` table. The `0` setting for the `error_number` parameter specifies all errors. The `ABEND` setting for the `method` parameter specifies that the inbound server is stopped when an error is encountered.

```
BEGIN
  DBMS_APPLY_ADM.SET_REPERROR_HANDLER(
    apply_name   => 'app_oe',
    object       => 'oe.orders',
    error_number => 0,
    method       => 'ABEND');
END;
/
```

Example 10-10 Setting an Error Handler That Ignores Row LCRs for a Specific Table and a Specific Error

This example sets an error handler that ignores row LCRs that raise the ORA-1403 error for the `app_oe` inbound server. The error handler applies to the `oe.orders` table.

```
BEGIN
  DBMS_APPLY_ADM.SET_REPERROR_HANDLER(
    apply_name   => 'app_oe',
    object       => 'oe.orders',
    error_number => 1403,
    method       => 'IGNORE');
END;
/
```

Example 10-11 Unsetting an Error Handler

This example unsets an error handler that ignores row LCRs that raise the ORA-1403 error for the `app_oe` inbound server. The error handler was set for the `oe.orders` table.

```
BEGIN
  DBMS_APPLY_ADM.SET_REPERROR_HANDLER(
    apply_name   => 'app_oe',
    object       => 'oe.orders',
    error_number => 1403,
    method       => NULL);
END;
/
```

Retrying Apply Error Transactions

You can retry a specific error transaction, or you can retry all error transactions for an inbound server.

Before you retry error transactions, you might need to make DML or DDL changes to database objects to correct the conditions that caused one or more apply errors.

- Retrying a Specific Apply Error Transaction
  When you retry an error transaction, you can execute it immediately or send the error transaction to a user procedure for modifications before executing it.
- Retrying All Error Transactions for an Inbound Server
  After you correct the conditions that caused all of the apply errors for an inbound server, you can retry all of the error transactions by running the `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.

Retrying a Specific Apply Error Transaction

When you retry an error transaction, you can execute it immediately or send the error transaction to a user procedure for modifications before executing it.

- Retrying a Specific Apply Error Transaction Without a User Procedure
  After you correct the conditions that caused an apply error, you can retry the transaction by running the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package without specifying a user procedure. In this case, the transaction executes without any custom processing.
- Retrying a Specific Apply Error Transaction With a User Procedure
  You can retry an error transaction by running the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package and specify a user procedure to modify one or more LCRs in the transaction before the transaction is executed.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `EXECUTE_ERROR` procedure

Retrying a Specific Apply Error Transaction Without a User Procedure

After you correct the conditions that caused an apply error, you can retry the transaction by running the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package without specifying a user procedure. In this case, the transaction executes without any custom processing.

When there are multiple error transactions, transaction ordering might be important when you execute them. In general, it is best practice to execute the oldest transaction first, and then each later transaction in order until you reach the newest transaction. The `SOURCE_COMMIT_POSITION` column in the `DBA_APPLY_ERROR` view shows the transaction time.

To retry a specific apply error transaction without a user 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 `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package, and specify the transaction identifier.

   To retry a transaction with the transaction identifier `5.4.312`, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ERROR(
    local_transaction_id => '5.4.312',
    execute_as_user      => FALSE,
    user_procedure       => NULL);
END;
/
```

If `execute_as_user` is `TRUE`, then the inbound server executes the transaction in the security context of the current user. If `execute_as_user` is `FALSE`, then the inbound server executes the transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised.

In either case, the user who executes the transaction must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers.

Retrying a Specific Apply Error Transaction With a User Procedure

You can retry an error transaction by running the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package and specify a user procedure to modify one or more LCRs in the transaction before the transaction is executed.

The modifications should enable successful execution of the transaction.

For example, consider a case in which a conflict caused an apply error. Examination of the error transaction reveals that the old value for the `salary` column in a row LCR contained the wrong value. Specifically, the current value of the salary of the employee with `employee_id` of `197` in the `hr.employees` table did not match the old value of the salary for this employee in the row LCR. Assume that the current value for this employee is `3250` in the `hr.employees` table. The example in this section creates a procedure to resolve the error.

To retry a specific apply error transaction with a user 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. Given this scenario described previously, create the following user procedure to modify the salary in the row LCR that caused the error:

```
CREATE OR REPLACE PROCEDURE xstrmadmin.modify_emp_salary(
  in_any                        IN      ANYDATA,
  error_record                  IN      ALL_APPLY_ERROR%ROWTYPE,
  error_message_number          IN      NUMBER,
  messaging_default_processing  IN OUT  BOOLEAN,
  out_any                       OUT     ANYDATA)
AS
  row_lcr          SYS.LCR$_ROW_RECORD;
  row_lcr_changed  BOOLEAN := FALSE;
  res              NUMBER;
  ob_owner         VARCHAR2(32);
  ob_name          VARCHAR2(32);
  cmd_type         VARCHAR2(30);
  employee_id      NUMBER;
BEGIN
  IF in_any.getTypeName() = 'SYS.LCR$_ROW_RECORD' THEN
    -- Access the LCR
    res := in_any.GETOBJECT(row_lcr);
    -- Determine the owner of the database object for the LCR
    ob_owner := row_lcr.GET_OBJECT_OWNER;
    -- Determine the name of the database object for the LCR
    ob_name := row_lcr.GET_OBJECT_NAME;
    -- Determine the type of DML change
    cmd_type := row_lcr.GET_COMMAND_TYPE;
    IF (ob_owner = 'HR' AND ob_name = 'EMPLOYEES' AND cmd_type = 'UPDATE') THEN
      -- Determine the employee_id of the row change
      IF row_lcr.GET_VALUE('old', 'employee_id') IS NOT NULL THEN
        employee_id := row_lcr.GET_VALUE('old', 'employee_id').ACCESSNUMBER();
        IF (employee_id = 197) THEN
          -- error_record.message_number should equal error_message_number
          row_lcr.SET_VALUE(
          value_type => 'OLD',
          column_name => 'salary',
          column_value => ANYDATA.ConvertNumber(3250));
          row_lcr_changed := TRUE;
        END IF;
      END IF;
    END IF;
  END IF;
  -- Specify that the inbound server continues to process the current message
  messaging_default_processing := TRUE;
  -- assign out_any appropriately
  IF row_lcr_changed THEN
    out_any := ANYDATA.ConvertObject(row_lcr);
  ELSE
    out_any := in_any;
  END IF;
END;
/
```

3. Run the EXECUTE_ERROR procedure in the DBMS_APPLY_ADM package, and specify the transaction identifier and the user procedure.

   To retry a transaction with the transaction identifier 5.6.924 and process the transaction with the modify_emp_salary procedure in the xstrmadmin schema before execution, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ERROR(
    local_transaction_id => '5.6.924',
    execute_as_user      => FALSE,
    user_procedure       => 'xstrmadmin.modify_emp_salary');
END;
/
```

Note:

The user who runs the procedure must have SELECT privilege on the ALL_APPLY_ERROR data dictionary view.

Retrying All Error Transactions for an Inbound Server

After you correct the conditions that caused all of the apply errors for an inbound server, you can retry all of the error transactions by running the EXECUTE_ALL_ERRORS procedure in the DBMS_APPLY_ADM package.

When there are multiple error transactions, the EXECUTE_ALL_ERRORS procedure executes the oldest transaction first, and then executes each later transaction in order up to the newest transaction.

To retry all error transactions for an inbound server:

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 EXECUTE_ALL_ERRORS procedure in the DBMS_APPLY_ADM package, and specify the name of the inbound server.

To retry all of the error transactions for an inbound server named `xin`, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
    apply_name       => 'xin',
    execute_as_user  => FALSE);
END;
/
```

Note:

If you specify `NULL` for the `apply_name` parameter, and you have multiple inbound servers, then all of the apply errors are retried for all of the inbound servers.

Deleting Apply Error Transactions

You can delete a specific error transaction, or you can delete all error transactions for an inbound server.

- Deleting a Specific Apply Error Transaction
  If an error transaction should not be applied, then you can delete the transaction from the error queue using the `DELETE_ERROR` procedure in the `DBMS_APPLY_ADM` package.
- Deleting All Error Transactions for an Inbound Server
  If none of the error transactions should be applied, then you can delete all of the error transactions by running the `DELETE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.

Deleting a Specific Apply Error Transaction

If an error transaction should not be applied, then you can delete the transaction from the error queue using the `DELETE_ERROR` procedure in the `DBMS_APPLY_ADM` package.

To delete a specific apply error transaction:

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. Identify the transaction ID of the error transaction you want to delete.

   For example, run the following query to list the local apply error transactions:

```
COLUMN APPLY_NAME FORMAT A11
COLUMN SOURCE_DATABASE' FORMAT A10
COLUMN LOCAL_TRANSACTION_ID FORMAT A11
COLUMN ERROR_NUMBER FORMAT 99999999
COLUMN ERROR_MESSAGE FORMAT A20
COLUMN MESSAGE_COUNT FORMAT 99999999

SELECT APPLY_NAME,
       SOURCE_DATABASE,
       LOCAL_TRANSACTION_ID,
       ERROR_NUMBER,
       ERROR_MESSAGE,
       MESSAGE_COUNT
  FROM DBA_APPLY_ERROR;
```

3. Run the `DELETE_ERROR` procedure in the `DBMS_APPLY_ADM` package, and specify the transaction identifier.

   To delete a transaction with the transaction identifier `5.4.312`, run the following procedure:

   ```
   EXEC DBMS_APPLY_ADM.DELETE_ERROR(local_transaction_id => '5.4.312');
   ```

Deleting All Error Transactions for an Inbound Server

If none of the error transactions should be applied, then you can delete all of the error transactions by running the `DELETE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.

To delete all error transactions for an inbound server:

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 `DELETE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package, and specify the name of the inbound server.

   To delete all of the error transactions for an inbound server named `xin`, run the following procedure:

   ```
   EXEC DBMS_APPLY_ADM.DELETE_ALL_ERRORS(apply_name => 'xin');
   ```

Note:

If you specify `NULL` for the `apply_name` parameter, and you have multiple inbound servers, then all of the apply errors are deleted for all of the inbound servers.

Managing Eager Errors Encountered by an Inbound Server

As a performance optimization, an inbound server can use eager apply to begin to apply large transactions before it receives the commit LCR.

See "Optimizing XStream In Performance for Large Transactions" for information about eager apply.

An inbound server can encounter an error while eagerly applying a transaction. Because all of the LCRs are not available for the transaction, an `EAGER ERROR` is recorded for this failed transaction. In this case, an entry in the `ALL_APPLY_ERROR` view shows an eager error for the transaction, but the LCRs are not recorded in the error queue. 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.

An eager error causes the inbound server to stop. When it restarts, if the error queue has an `EAGER ERROR` for the restarting transaction, then the transaction is started as a normal transaction. That is, the LCRs in the large transaction spill to disk, and the inbound server begins to apply them only after the commit LCR is received.

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

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

Table 10-3 explains the options for managing a normal error transaction.

Table 10-3 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 Cloud Control
		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 Cloud Control
		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 10-4 explains the options for managing an eager error transaction.

Table 10-4 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 Cloud Control
		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.

Table 10-4 (Cont.) Options Available for Managing an Eager Error Transaction

Action	Mechanisms	Description
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 Cloud Control, 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 "Retrying Apply Error Transactions" and "Deleting Apply Error Transactions".

3. Examine the error message raised by the LCR, and determine the cause of the error.

> **See Also:**
>
> - "Checking for Apply Errors" and "Displaying Detailed Information About Apply Errors" for information about checking for apply errors using data dictionary views
> - Oracle Enterprise Manager Cloud Control online help for information about checking for apply errors using Oracle Enterprise Manager Cloud Control

4. If possible, determine how to avoid the error, and make any changes necessary to avoid the error.

> **See Also:**
>
> "Troubleshooting XStream In" for information about common apply errors and solutions for them

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.

     For example, to delete a transaction with the transaction identifier `5.4.312`, run the following procedure:

     ```
     EXEC DBMS_APPLY_ADM.DELETE_ERROR(local_transaction_id => '5.4.312');
     ```
   - Retain the error transaction if you cannot correct the problem now or if you plan to reexecute it in the future. No action is necessary to retain the error transaction. It remains in the error queue until it is reexecuted or deleted.

   See Table 10-4 for more information about these choices.

> **Note:**
>
> 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 Also:**
>
> - "Deleting Apply Error Transactions" for more information about deleting an error transaction using the `DBMS_APPLY_ADM` package
> - See the Oracle Enterprise Manager Cloud Control online help for information about deleting an error transaction using Oracle Enterprise Manager Cloud Control.

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 10-4 for information about how the inbound server handles the error transaction based on your choice in Step 5.

   See Also:

   - "Starting an Inbound Server" for information about starting an inbound server or apply process using the `DBMS_APPLY_ADM` package
   - Oracle Enterprise Manager Cloud Control online help for information about starting an inbound server or apply process using Oracle Enterprise Manager Cloud Control

See Also:

- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*

Conflict and Error Handling Precedence

To resolve a conflict or error, an inbound server tries to find conflict handlers and error handlers.

When an inbound server encounters a conflict or an error, it tries to resolve the problem by checking for the following types of handlers that apply to the error in the specified order:

1. An update conflict handler set with the `SET_UPDATE_CONFLICT_HANDLER` procedure
2. A custom conflict handler set with the `SET_DML_CONFLICT_HANDLER` procedure
3. A collision handler set with the `HANDLE_COLLISIONS` procedure
4. An error handler set with the `SET_REPERROR_HANDLER` procedure
5. A custom conflict handler set with the `SET_DML_HANDLER` procedure

All of the procedures are in the `DBMS_APPLY_ADM` package.

If no handler applies to the conflict or error, then the transaction that caused the error is moved to the error queue.

Dropping Components in an XStream In Configuration

You can drop an inbound server with 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.

If the inbound server's queue is not dropped automatically, then run the `REMOVE_QUEUE` procedure to drop it.

Example 10-12 Dropping an Inbound Server

To drop an inbound server named `xin`, run the following procedure:

```
exec DBMS_XSTREAM_ADM.DROP_INBOUND('xin');
```

Example 10-13 Dropping an Inbound Server's Queue

To drop a queue named `xin_queue`, run the following procedure:

```
exec DBMS_XSTREAM_ADM.REMOVE_QUEUE('xin_queue');
```

See Also:

Oracle Database PL/SQL Packages and Types Reference

11

Monitoring XStream In

You can monitor an XStream In configuration by querying data dictionary views.

This chapter provides instructions for monitoring XStream.

With XStream In, an Oracle Apply process functions as an inbound server. Therefore, you can also use the data dictionary views for apply processes to monitor inbound servers.

Note:

Whenever possible, this chapter uses ALL_ static data dictionary views for query examples. In some cases, information in the ALL_ views is more limited than the information in the DBA_ views.

In SQL*Plus, trusted XStream administrators can query the ALL_ and DBA_ views. Untrusted XStream administrators can query the ALL_ views only.

- Displaying Session Information for Inbound Servers
  An example illustrates displaying session information for inbound servers.
- Displaying General Information About an Inbound Server
  An example illustrates displaying general information about an inbound server.
- Monitoring the History of Events for XStream In Components
  An example illustrates monitoring the history of events for XStream In components by querying the DBA_REPLICATION_PROCESS_EVENTS view.
- Displaying the Status and Error Information for an Inbound Server
  An example illustrates displaying the status and error information for an inbound server.
- Displaying Apply Parameter Settings for an Inbound Server
  An example illustrates displaying apply parameter settings for an inbound server.
- Displaying the Position Information for an Inbound Server
  An example illustrates displaying the position information for an inbound server.
- Displaying Information About DML Conflict Handlers
  The DBA_APPLY_DML_CONF_HANDLERS view displays information about DML conflict handlers.
- Displaying Information About Error Handlers
  The DBA_APPLY_REPERROR_HANDLERS view displays information about DML conflict handlers.
- Checking for Apply Errors
  An example illustrates checking for apply errors.
- Displaying Detailed Information About Apply Errors
  SQL scripts display detailed information about the error transactions in the error queue in a database.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- "Troubleshooting XStream In"

Displaying Session Information for Inbound Servers

An example illustrates displaying session information for inbound servers.

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 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 In configuration looks similar to the following:

```
                                            Session                   XStream
                                             Serial Operating System  Process
XStream Component              Session ID    Number Process Number    Number
------------------------------ ---------- --------- ----------------- -------
XIN - Apply Reader                     19         9 27304             AS01
XIN - Apply Server                     22         5 27308             AS03
XIN - Apply Server                     25        31 27313             AS05
XIN - Apply Coordinator               112         7 27302             AP01
XIN - Apply Server                    113          5 27306              AS02
XIN - Propagation Receiver            114         17 27342              TNS
XIN - Apply Server                    115         39 27311              AS04
```

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.

Note:

To run this query, a user must have the necessary privileges to query the `V$SESSION` view.

See Also:

Oracle Database Reference for more information about the `V$SESSION` view

Displaying General Information About an Inbound Server

An example illustrates 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 ALL_XSTREAM_INBOUND;
   ```

Your output looks similar to the following:

```
Inbound Server Name  Queue Owner     Queue Name      Apply User
-------------------- --------------- --------------- ---------------
XIN                  XSTRMADMIN      XIN_QUEUE       XSTRMADMIN
```

See Also:

Oracle Database Reference

Monitoring the History of Events for XStream In Components

An example illustrates monitoring the history of events for XStream In components by querying the `DBA_REPLICATION_PROCESS_EVENTS` view.

For example, this view can display when a component was created or started. It can also display when a component parameter was changed. If the component encountered an error, then it can display information about the error.

The query in this topic displays the following information about XStream Out component events:

- The XStream component name
- The component type
- The event name
- The description of the event
- The event time

To display this information for each XStream In 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 STREAMS_NAME FORMAT A12
   COLUMN PROCESS_TYPE FORMAT A17
   COLUMN EVENT_NAME FORMAT A10
   COLUMN DESCRIPTION FORMAT A20
   COLUMN EVENT_TIME FORMAT A15

   SELECT STREAMS_NAME,
          PROCESS_TYPE,
          EVENT_NAME,
          DESCRIPTION,
          EVENT_TIME
     FROM DBA_REPLICATION_PROCESS_EVENTS;
   ```

Your output for an XStream In configuration looks similar to the following:

```
STREAMS_NAME PROCESS_TYPE      EVENT_NAME DESCRIPTION          EVENT_TIME
------------ ----------------- ---------- -------------------- ---------------
APP_JOBS     APPLY COORDINATOR CREATE     SUCCESS              03-NOV-15 07.19
                                                               .27.238151 AM
APP_JOBS     APPLY COORDINATOR START      SUCCESS              03-NOV-15 07.21
```

```
                                                              .50.812534 AM
APP_JOBS     APPLY READER      START      SUCCESS             03-NOV-15 07.21
                                                              .51.713367 AM
APP_JOBS     APPLY SERVER      START      SUCCESS             03-NOV-15 07.21
                                                              .51.895019 AM
```

Related Topics

- *Oracle Database Reference*

Displaying the Status and Error Information for an Inbound Server

An example illustrates displaying the status and error information for an inbound server.

The `DBA_APPLY` view shows `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 'Inbound Server|Name' FORMAT A15
   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:

```
Inbound Server
Name            Status   Error Number Error Message
--------------- -------- ------------ ----------------------------------------
XIN             ENABLED
```

This output shows that `XIN` is an apply process that is functioning as an inbound server.

Note:

This example queries the `DBA_APPLY` view. This view enables trusted users to see information for all apply users in the database. Untrusted users must query the `ALL_APPLY` view, which limits information to the current user.

See Also:

Oracle Database Reference

Displaying Apply Parameter Settings for an Inbound Server

An example illustrates displaying apply parameter settings for an inbound server.

Apply parameters determine how an inbound server operates. To display the apply parameter settings 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 APPLY_NAME HEADING 'Inbound Server|Name' FORMAT A15
COLUMN PARAMETER HEADING 'Parameter' FORMAT A30
COLUMN VALUE HEADING 'Value' FORMAT A22
COLUMN SET_BY_USER HEADING 'Set by|User?' FORMAT A10

SELECT APPLY_NAME,
       PARAMETER,
       VALUE,
       SET_BY_USER
  FROM ALL_APPLY_PARAMETERS a, ALL_XSTREAM_INBOUND i
  WHERE a.APPLY_NAME=i.SERVER_NAME
  ORDER BY a.PARAMETER;
```

Your output looks similar to the following:

```
Inbound Server                                                       Set by
Name            Parameter                      Value                  User?
--------------- ------------------------------ ---------------------- ----------
XIN             ALLOW_DUPLICATE_ROWS           N                      NO
XIN             APPLY_SEQUENCE_NEXTVAL         Y                      NO
XIN             COMMIT_SERIALIZATION           DEPENDENT_TRANSACTIONS NO
XIN             COMPARE_KEY_ONLY               Y                      NO
XIN             COMPUTE_LCR_DEP_ON_ARRIVAL     N                      NO
XIN             DISABLE_ON_ERROR               Y                      NO
XIN             DISABLE_ON_LIMIT               N                      NO
XIN             EAGER_SIZE                     9500                   NO
XIN             ENABLE_XSTREAM_TABLE_STATS     Y                      NO
XIN             EXCLUDETAG                                            NO
XIN             EXCLUDETRANS                                          NO
XIN             EXCLUDEUSER                                           NO
XIN             EXCLUDEUSERID                                         NO
XIN             GETAPPLOPS                     Y                      NO
XIN             GETREPLICATES                  N                      NO
XIN             GROUPTRANSOPS                  250                    NO
XIN             HANDLECOLLISIONS               N                      NO
XIN             IGNORE_TRANSACTION                                    NO
XIN             MAXIMUM_SCN                    INFINITE               NO
XIN             MAX_PARALLELISM                50                     NO
XIN             MAX_SGA_SIZE                   INFINITE               NO
XIN             OPTIMIZE_PROGRESS_TABLE        N                      NO
XIN             OPTIMIZE_SELF_UPDATES          Y                      NO
XIN             PARALLELISM                    4                      NO
XIN              PRESERVE_ENCRYPTION            Y                       NO
XIN              RTRIM_ON_IMPLICIT_CONVERSION   Y                       NO
XIN              STARTUP_SECONDS                0                       NO
XIN              SUPPRESSTRIGGERS               Y                       NO
XIN              TIME_LIMIT                     INFINITE                NO
XIN              TRACE_LEVEL                    0                       NO
XIN              TRANSACTION_LIMIT              INFINITE                NO
XIN              TXN_AGE_SPILL_THRESHOLD        900                     NO
XIN              TXN_LCR_SPILL_THRESHOLD        10000                   NO
XIN              WRITE_ALERT_LOG                Y                       NO
```

Inbound servers ignore some apply parameter settings.

Note:

If the `SET_BY_USER` column is `NO` for a parameter, then the parameter is set to its default value. If the `SET_BY_USER` column is `YES` for a parameter, then the parameter was set by a user and might or might not be set to its default value.

Desupport of OPTIMIZE_PROGRESS_TABLE Parameter

`OPTIMIZE_PROGRESS_TABLE` for Oracle GoldenGate Integrated Replicat, XStream In, and Logical Standby, is desupported in Oracle Database 19c
The apply parameter `OPTIMIZE_PROGRESS_TABLE` for Oracle GoldenGate Integrated Replicat, XStream In, and Logical Standby was desupported in Oracle Database 19c. Before you upgrade to Oracle Database 19c, you must turn off this parameter. If `OPTIMIZE_PROGRESS_TABLE` set to `ON`, then stop apply gracefully, turn off the parameter, and restart apply. For GoldenGate Apply and XStream, this parameter was set to `OFF`, by default

See Also:

- "Setting an Apply Parameter for an Inbound Server"
- *Oracle Database PL/SQL Packages and Types Reference* for information about apply parameters

Displaying the Position Information for an Inbound Server

An example illustrates displaying the position information for an inbound server.

For an inbound server, you can view position information by querying the `ALL_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 ALL_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:

- *Oracle Database Reference*
- "Position Order in an LCR Stream"
- "Position of LCRs and XStream In"

Displaying Information About DML Conflict Handlers

The DBA_APPLY_DML_CONF_HANDLERS view displays information about DML conflict handlers.

You can configure DML conflict handlers using the SET_DML_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package.

1. Connect to the database as the XStream administrator.
2. Query the DBA_APPLY_DML_CONF_HANDLERS view.

Example 11-1 Displaying Information About DML Conflict Handlers

```
COLUMN APPLY_NAME FORMAT A8
COLUMN OBJECT_OWNER FORMAT A5
COLUMN OBJECT_NAME FORMAT A12
COLUMN COMMAND_TYPE FORMAT A6
COLUMN CONFLICT_TYPE FORMAT A11
COLUMN METHOD_NAME FORMAT A12
COLUMN CONFLICT_HANDLER_NAME FORMAT A20

SELECT APPLY_NAME,
       OBJECT_OWNER,
       OBJECT_NAME,
       COMMAND_TYPE,
       CONFLICT_TYPE,
       METHOD_NAME,
       CONFLICT_HANDLER_NAME
  FROM DBA_APPLY_DML_CONF_HANDLERS
  ORDER BY OBJECT_OWNER, OBJECT_NAME, CONFLICT_HANDLER_NAME;
```

Your output looks similar to the following:

```
APPLY_NA OBJEC OBJECT_NAME  COMMAN CONFLICT_TY METHOD_NAME
CONFLICT_HANDLER_NAM
-------- ----- ------------ ------ ----------- ------------
--------------------
APP_JOBS HR    JOBS         DELETE ROW_MISSING IGNORE
JOBS_HANDLER_DELETE
APP_JOBS HR    JOBS         INSERT ROW_EXISTS  OVERWRITE
JOBS_HANDLER_INSERT
APP_JOBS HR    JOBS         UPDATE ROW_EXISTS  OVERWRITE
JOBS_HANDLER_UPDATE
```

Related Topics

- Setting a DML Conflict Handler
  Set a DML conflict handler using the `SET_DML_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package.

Displaying Information About Error Handlers

The `DBA_APPLY_REPERROR_HANDLERS` view displays information about DML conflict handlers.

You can configure error handlers using the `SET_REPERROR_HANDLER` procedure in the `DBMS_APPLY_ADM` package.

1. Connect to the database as the XStream administrator.
2. Query the `DBA_APPLY_REPERROR_HANDLERS` view.

Example 11-2   Displaying Information About DML Conflict Handlers

```
COLUMN APPLY_NAME FORMAT A15
COLUMN OBJECT_OWNER FORMAT A15
COLUMN OBJECT_NAME FORMAT A15
COLUMN ERROR_NUMBER 999999999
COLUMN METHOD FORMAT A15

SELECT APPLY_NAME,
       OBJECT_OWNER,
       OBJECT_NAME,
       ERROR_NUMBER,
       METHOD
  FROM DBA_APPLY_REPERROR_HANDLERS
  ORDER BY OBJECT_OWNER, OBJECT_NAME;
```

Your output looks similar to the following:

```
APPLY_NAME      OBJECT_OWNER    OBJECT_NAME     ERROR_NUMBER METHOD
--------------- --------------- --------------- ------------ ---------------
APP_OE          OE              ORDERS                 26787 IGNORE
```

Related Topics

- Setting and Unsetting an Error Handler
  You set an error handler with the `SET_REPERROR_HANDLER` procedure in the `DBMS_APPLY` package.

Checking for Apply Errors

An example illustrates checking for apply errors.

Trusted users can check for apply errors by querying the `DBA_APPLY_ERROR` data dictionary view or by using Oracle Enterprise Manager Cloud Control. Untrusted users can check for apply errors by querying the `ALL_APPLY_ERROR` data dictionary view.

To check for apply errors:

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 'Inbound|Server|Name' FORMAT A7
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A8
COLUMN SOURCE_TRANSACTION_ID HEADING 'Source|Transaction|ID' FORMAT A11
COLUMN MESSAGE_NUMBER HEADING 'Failed Message|in Error|Transaction' FORMAT 99999999
COLUMN ERROR_NUMBER HEADING 'Error Number' FORMAT 99999999
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A10
COLUMN MESSAGE_COUNT HEADING 'Messages in|Error|Transaction' FORMAT 99999999

SELECT APPLY_NAME,
       SOURCE_DATABASE,
       SOURCE_TRANSACTION_ID,
       MESSAGE_NUMBER,
       ERROR_NUMBER,
       ERROR_MESSAGE,
       MESSAGE_COUNT
  FROM ALL_APPLY_ERROR;
```

Note:

Trusted users should replace `ALL_APPLY_ERROR` with `DBA_APPLY_ERROR` in the query.

If there are any apply errors, then your output looks similar to the following:

```
Inbound          Source      Failed Message                            Messages in
Server  Source   Transaction       in Error                                  Error
Name    Database ID             Transaction Error Number Error Mess Transaction
------- -------- ----------- -------------- ------------ ---------- -----------
XIN     OUTX.EXA 19.20.215                1         1031 ORA-01031:           1
        MPLE.COM                                          insuffici
                                                         ent privil
                                                         eges
XIN     OUTX.EXA 11.21.158                1         1031 ORA-01031:           1
        MPLE.COM                                          insuffici
                                                         ent privil
                                                         eges
```

If there are apply errors, then you can either try to reexecute the transactions that encountered the errors, or you can delete the transactions. To reexecute a transaction that encountered an error, first correct the condition that caused the transaction to raise an error.

If you want to delete a transaction that encountered an error, and if you are sharing data between multiple databases, then you might need to resynchronize data manually. Remember to set an appropriate session tag, if necessary, when you resynchronize data manually.

See Also:

- "The Error Queue for an Inbound Server"
- "Managing Apply Errors"

Displaying Detailed Information About Apply Errors

SQL scripts display detailed information about the error transactions in the error queue in a database.

- Step 1: Grant Explicit SELECT Privilege on the ALL_APPLY_ERROR View
- Step 2: Create a Procedure that Prints the Value in an ANYDATA Object
  Create a procedure that prints the value in a specified `ANYDATA` object for some selected data types. Optionally, you can add more data types to this procedure.
- Step 3: Create a Procedure that Prints a Specified LCR
  Create a procedure that prints a specified LCR.
- Step 4: Create a Procedure that Prints All the LCRs in the Error Queue
  Create a procedure that prints all of the LCRs in all of the error queues.
- Step 5: Create a Procedure that Prints All the Error LCRs for a Transaction
  Create a procedure that prints all the LCRs in the error queue for a particular transaction.

See Also:

- "The Error Queue for an Inbound Server"
- "Managing Apply Errors"

Step 1: Grant Explicit SELECT Privilege on the ALL_APPLY_ERROR View

The user who creates and runs the `print_errors` and `print_transaction` procedures described in the following sections must be granted explicit `SELECT` privilege on the `ALL_APPLY_ERROR` data dictionary view. This privilege cannot be granted through a role because the procedures in the following section are definer's rights.

To grant explicit SELECT privilege on the `ALL_APPLY_ERROR` view:

1. In SQL*Plus, connect as an administrative user who can grant privileges.

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

2. Grant `SELECT` privilege on the `ALL_APPLY_ERROR` data dictionary view to the appropriate user. For example, to grant this privilege to the `xstrmadmin` user, run the following statement:

   ```
   GRANT SELECT ON ALL_APPLY_ERROR TO xstrmadmin;
   ```

3. Grant `EXECUTE` privilege on the `DBMS_APPLY_ADM` package. For example, to grant this privilege to the `xstrmadmin` user, run the following statement:

   ```
   GRANT EXECUTE ON DBMS_APPLY_ADM TO xstrmadmin;
   ```

4. Connect to the database as the user to whom you granted the privilege in Step 2 and 3.

Step 2: Create a Procedure that Prints the Value in an ANYDATA Object

Create a procedure that prints the value in a specified `ANYDATA` object for some selected data types. Optionally, you can add more data types to this procedure.

```
CREATE OR REPLACE PROCEDURE print_any(data IN ANYDATA) IS
  tn  VARCHAR2(61);
  str VARCHAR2(4000);
  chr VARCHAR2(1000);
  num NUMBER;
  dat DATE;
  rw  RAW(4000);
  res NUMBER;
BEGIN
  IF data IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('NULL value');
    RETURN;
  END IF;
  tn := data.GETTYPENAME();
  IF tn = 'SYS.VARCHAR2' THEN
    res := data.GETVARCHAR2(str);
    DBMS_OUTPUT.PUT_LINE(SUBSTR(str,0,253));
  ELSIF tn = 'SYS.CHAR' then
    res := data.GETCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(SUBSTR(chr,0,253));
```

```
  ELSIF tn = 'SYS.VARCHAR' THEN
    res := data.GETVARCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(chr);
  ELSIF tn = 'SYS.NUMBER' THEN
    res := data.GETNUMBER(num);
    DBMS_OUTPUT.PUT_LINE(num);
  ELSIF tn = 'SYS.DATE' THEN
    res := data.GETDATE(dat);
    DBMS_OUTPUT.PUT_LINE(dat);
  ELSIF tn= 'SYS.TIMESTAMP' THEN
    res := data.GETTIMESTAMP(dat);
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(dat,'DD-MON-RR HH.MI.SSXFF AM'));
  ELSIF tn= 'SYS.TIMESTAMPTZ' THEN
    res := data.GETTIMESTAMPTZ(dat);
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(dat,'DD-MON-RR HH.MI.SSXFF AM'));
  ELSIF tn= 'SYS.TIMESTAMPLTZ' THEN
    res := data.GETTIMESTAMPLTZ(dat);
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(dat,'DD-MON-RR HH.MI.SSXFF AM'));
  ELSIF tn = 'SYS.RAW' THEN
    -- res := data.GETRAW(rw);
    -- DBMS_OUTPUT.PUT_LINE(SUBSTR(DBMS_LOB.SUBSTR(rw),0,253));
    DBMS_OUTPUT.PUT_LINE('BLOB Value');
  ELSIF tn = 'SYS.BLOB' THEN
    DBMS_OUTPUT.PUT_LINE('BLOB Found');
  ELSE
    DBMS_OUTPUT.PUT_LINE('typename is ' || tn);
  END IF;
END print_any;
/
```

Step 3: Create a Procedure that Prints a Specified LCR

Create a procedure that prints a specified LCR.

The procedure calls the `print_any` procedure created in "Step 2: Create a Procedure that Prints the Value in an ANYDATA Object".

```
CREATE OR REPLACE PROCEDURE print_lcr(lcr IN ANYDATA) IS
  typenm    VARCHAR2(61);
  ddllcr    SYS.LCR$_DDL_RECORD;
  proclcr   SYS.LCR$_PROCEDURE_RECORD;
  rowlcr    SYS.LCR$_ROW_RECORD;
  res       NUMBER;
  newlist   SYS.LCR$_ROW_LIST;
  oldlist   SYS.LCR$_ROW_LIST;
  ddl_text  CLOB;
  ext_attr  ANYDATA;
BEGIN
  typenm := lcr.GETTYPENAME();
  DBMS_OUTPUT.PUT_LINE('type name: ' || typenm);
  IF (typenm = 'SYS.LCR$_DDL_RECORD') THEN
    res := lcr.GETOBJECT(ddllcr);
    DBMS_OUTPUT.PUT_LINE('source database: ' ||
                         ddllcr.GET_SOURCE_DATABASE_NAME);
    DBMS_OUTPUT.PUT_LINE('owner: ' || ddllcr.GET_OBJECT_OWNER);
    DBMS_OUTPUT.PUT_LINE('object: ' || ddllcr.GET_OBJECT_NAME);
    DBMS_OUTPUT.PUT_LINE('is tag null: ' || ddllcr.IS_NULL_TAG);
    DBMS_LOB.CREATETEMPORARY(ddl_text, TRUE);
    ddllcr.GET_DDL_TEXT(ddl_text);
    DBMS_OUTPUT.PUT_LINE('ddl: ' || ddl_text);
    -- Print extra attributes in DDL LCR
    ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('serial#');
```

```
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('serial#: ' || ext_attr.ACCESSNUMBER());
        END IF;
      ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('session#');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('session#: ' || ext_attr.ACCESSNUMBER());
        END IF;
      ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('thread#');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('thread#: ' || ext_attr.ACCESSNUMBER());
        END IF;
      ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('tx_name');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('transaction name: ' || ext_attr.ACCESSVARCHAR2());
        END IF;
      ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('username');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('username: ' || ext_attr.ACCESSVARCHAR2());
        END IF;
      DBMS_LOB.FREETEMPORARY(ddl_text);
    ELSIF (typenm = 'SYS.LCR$_ROW_RECORD') THEN
      res := lcr.GETOBJECT(rowlcr);
      DBMS_OUTPUT.PUT_LINE('source database: ' ||
                           rowlcr.GET_SOURCE_DATABASE_NAME);
      DBMS_OUTPUT.PUT_LINE('owner: ' || rowlcr.GET_OBJECT_OWNER);
      DBMS_OUTPUT.PUT_LINE('object: ' || rowlcr.GET_OBJECT_NAME);
      DBMS_OUTPUT.PUT_LINE('is tag null: ' || rowlcr.IS_NULL_TAG);
      DBMS_OUTPUT.PUT_LINE('command_type: ' || rowlcr.GET_COMMAND_TYPE);
      oldlist := rowlcr.GET_VALUES('old');
      FOR i IN 1..oldlist.COUNT LOOP
        IF oldlist(i) IS NOT NULL THEN
          DBMS_OUTPUT.PUT_LINE('old(' || i || '): ' || oldlist(i).column_name);
          print_any(oldlist(i).data);
        END IF;
      END LOOP;
      newlist := rowlcr.GET_VALUES('new', 'n');
      FOR i in 1..newlist.count LOOP
        IF newlist(i) IS NOT NULL THEN
          DBMS_OUTPUT.PUT_LINE('new(' || i || '): ' || newlist(i).column_name);
          print_any(newlist(i).data);
        END IF;
      END LOOP;
      -- Print extra attributes in row LCR
      ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('row_id');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('row_id: ' || ext_attr.ACCESSUROWID());
        END IF;
      ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('serial#');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('serial#: ' || ext_attr.ACCESSNUMBER());
        END IF;
      ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('session#');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('session#: ' || ext_attr.ACCESSNUMBER());
        END IF;
      ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('thread#');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('thread#: ' || ext_attr.ACCESSNUMBER());
        END IF;
      ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('tx_name');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('transaction name: ' || ext_attr.ACCESSVARCHAR2());
        END IF;
      ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('username');
        IF (ext_attr IS NOT NULL) THEN
          DBMS_OUTPUT.PUT_LINE('username: ' || ext_attr.ACCESSVARCHAR2());
        END IF;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Non-LCR Message with type ' || typenm);
    END IF;
  END print_lcr;
  /
```

Step 4: Create a Procedure that Prints All the LCRs in the Error Queue

Create a procedure that prints all of the LCRs in all of the error queues.

The procedure calls the `print_lcr` procedure created in "Step 3: Create a Procedure that Prints a Specified LCR".

```
CREATE OR REPLACE PROCEDURE print_errors IS
  CURSOR c IS
    SELECT LOCAL_TRANSACTION_ID,
           SOURCE_DATABASE,
           MESSAGE_NUMBER,
           MESSAGE_COUNT,
           ERROR_NUMBER,
           ERROR_MESSAGE
      FROM ALL_APPLY_ERROR
      ORDER BY SOURCE_DATABASE, SOURCE_COMMIT_SCN;
  i      NUMBER;
  txnid  VARCHAR2(30);
  source VARCHAR2(128);
  msgno  NUMBER;
  msgcnt NUMBER;
  errnum NUMBER := 0;
  errno  NUMBER;
  errmsg VARCHAR2(2000);
  lcr    ANYDATA;
  r      NUMBER;
BEGIN
  FOR r IN c LOOP
    errnum := errnum + 1;
    msgcnt := r.MESSAGE_COUNT;
    txnid  := r.LOCAL_TRANSACTION_ID;
    source := r.SOURCE_DATABASE;
    msgno  := r.MESSAGE_NUMBER;
    errno  := r.ERROR_NUMBER;
    errmsg := r.ERROR_MESSAGE;
DBMS_OUTPUT.PUT_LINE('*************************************************');
    DBMS_OUTPUT.PUT_LINE('----- ERROR #' || errnum);
    DBMS_OUTPUT.PUT_LINE('----- Local Transaction ID: ' || txnid);
    DBMS_OUTPUT.PUT_LINE('----- Source Database: ' || source);
    DBMS_OUTPUT.PUT_LINE('----Error in Message: '|| msgno);
    DBMS_OUTPUT.PUT_LINE('----Error Number: '||errno);
    DBMS_OUTPUT.PUT_LINE('----Message Text: '||errmsg);
    FOR i IN 1..msgcnt LOOP
      DBMS_OUTPUT.PUT_LINE('--message: ' || i);
        lcr := DBMS_APPLY_ADM.GET_ERROR_MESSAGE(i, txnid);
        print_lcr(lcr);
    END LOOP;
  END LOOP;
END print_errors;
/
```

To run this procedure after you create it, enter the following:

```
SET SERVEROUTPUT ON SIZE 1000000

EXEC print_errors
```

Step 5: Create a Procedure that Prints All the Error LCRs for a Transaction

Create a procedure that prints all the LCRs in the error queue for a particular transaction.

The procedure calls the `print_lcr` procedure created in "Step 3: Create a Procedure that Prints a Specified LCR".

```
CREATE OR REPLACE PROCEDURE print_transaction(ltxnid IN VARCHAR2) IS
  i      NUMBER;
  txnid  VARCHAR2(30);
  source VARCHAR2(128);
  msgno  NUMBER;
  msgcnt NUMBER;
  errno  NUMBER;
  errmsg VARCHAR2(2000);
  lcr    ANYDATA;
BEGIN
  SELECT LOCAL_TRANSACTION_ID,
         SOURCE_DATABASE,
         MESSAGE_NUMBER,
         MESSAGE_COUNT,
         ERROR_NUMBER,
         ERROR_MESSAGE
      INTO txnid, source, msgno, msgcnt, errno, errmsg
      FROM ALL_APPLY_ERROR
      WHERE LOCAL_TRANSACTION_ID =  ltxnid;
  DBMS_OUTPUT.PUT_LINE('----- Local Transaction ID: ' || txnid);
  DBMS_OUTPUT.PUT_LINE('----- Source Database: ' || source);
  DBMS_OUTPUT.PUT_LINE('----Error in Message: '|| msgno);
  DBMS_OUTPUT.PUT_LINE('----Error Number: '||errno);
  DBMS_OUTPUT.PUT_LINE('----Message Text: '||errmsg);
  FOR i IN 1..msgcnt LOOP
  DBMS_OUTPUT.PUT_LINE('--message: ' || i);
    lcr := DBMS_APPLY_ADM.GET_ERROR_MESSAGE(i, txnid); -- gets the LCR
    print_lcr(lcr);
  END LOOP;
END print_transaction;
/
```

To run this procedure after you create it, pass to it the local transaction identifier of an error transaction. For example, if the local transaction identifier is `1.17.2485`, then enter the following:

```
SET SERVEROUTPUT ON SIZE 1000000

EXEC print_transaction('1.17.2485')
```

12
Troubleshooting XStream In

You can diagnose and correct problems with an XStream In configuration.

- Diagnosing Problems with XStream In
  You can diagnose problems with XStream In by using several different techniques.
- Problems and Solutions for XStream In
  You can implement solutions for common problems with XStream In.
- How to Get More Help with XStream In
  Oracle Support can provide more help with XStream In.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- "Configuring XStream Out"

Diagnosing Problems with XStream In

You can diagnose problems with XStream In by using several different techniques.

- Viewing Alerts
  An **alert** is a warning about a potential problem or an indication that a critical threshold has been crossed.
- Checking the Trace File and Alert Log for Problems
  Messages about inbound server are recorded in trace files for the database in which the process is running.

Viewing Alerts

An **alert** is a warning about a potential problem or an indication that a critical threshold has been crossed.

There are two types of alerts:

- **Stateless:** Alerts that indicate single events that are not necessarily tied to the system state. For example, an alert that indicates that a capture aborted with a specific error is a stateless alert.
- **Stateful:** Alerts that are associated with a specific system state. Stateful alerts are usually based on a numeric value, with thresholds defined at warning and critical levels. For example, an alert on the current Streams pool memory usage percentage, with the warning level at 85% and the critical level at 95%, is a stateful alert.

An Oracle database generates a stateless alert when an inbound server aborts.

An Oracle database generates a stateful XStream alert when the Streams pool memory usage exceeds the percentage specified by the `STREAMS_POOL_USED_PCT` metric. You can manage this metric with the `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package.

You can view alerts in Oracle Enterprise Manager Cloud Control, or you can query the following data dictionary views:

- The `DBA_OUTSTANDING_ALERTS` view records current stateful alerts. The `DBA_ALERT_HISTORY` view records stateless alerts and stateful alerts that have been cleared. For example, if the memory usage in the Streams pool exceeds the specified threshold, then a stateful alert is recorded in the `DBA_OUTSTANDING_ALERTS` view.
- The `DBA_ALERT_HISTORY` data dictionary view shows alerts that have been cleared from the `DBA_OUTSTANDING_ALERTS` view. For example, if the memory usage in the Streams pool falls below the specified threshold, then the alert recorded in the `DBA_OUTSTANDING_ALERTS` view is cleared and moved to the `DBA_ALERT_HISTORY` view.

For example, to list the current stateful alerts, run the following query on the `DBA_OUTSTANDING_ALERTS` view:

```
COLUMN REASON HEADING 'Reason for Alert' FORMAT A35
COLUMN SUGGESTED_ACTION HEADING 'Suggested Response' FORMAT A35

SELECT REASON, SUGGESTED_ACTION
   FROM DBA_OUTSTANDING_ALERTS
   WHERE MODULE_ID LIKE '%XSTREAM%';
```

To list the stateless alerts and cleared XStream stateful alerts, run the following query on the `DBA_ALERT_HISTORY` view:

```
COLUMN REASON HEADING 'Reason for Alert' FORMAT A35
COLUMN SUGGESTED_ACTION HEADING 'Suggested Response' FORMAT A35

SELECT REASON, SUGGESTED_ACTION
   FROM DBA_ALERT_HISTORY
   WHERE MODULE_ID LIKE '%XSTREAM%';
```

See Also:

- *Oracle Database Get Started with Performance Tuning* for information about managing alerts and metric thresholds
- *Oracle Database Administrator’s Guide* for information about alerts and for information about subscribing to the `ALERT_QUE` queue to receive notifications when new alerts are generated
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SERVER_ALERT` package
- "Configure the Streams pool"

Checking the Trace File and Alert Log for Problems

Messages about inbound server are recorded in trace files for the database in which the process is running.

These trace file messages can help you to identify and resolve problems in an XStream In configuration.

All trace files for background processes are written to the Automatic Diagnostic Repository. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file.

For example, on some operating systems, the trace file name for a process is `sid_xxxx_iiiii.trc`, where:

- *`sid`* is the system identifier for the database
- *`xxxx`* is the name of the process
- *`iiiii`* is the operating system process number

Also, you can set the `write_alert_log` parameter to `y` for both a capture process and an outbound server. When this parameter is set to `y`, which is the default setting, the alert log for the database contains messages about why the capture process or outbound server stopped.

You can control the information in the trace files by setting the `trace_level` inbound server apply parameter using the `SET_PARAMETER` procedure in the `DBMS_XSTREAM_ADM` package.

An inbound server is an Oracle background process named `AP`*`nn`*, where *`nn`* can include letters and numbers. For example, on some operating systems, if the system identifier for a database running an inbound server is `hqdb` and the inbound server number is `01`, then the trace file for the inbound server starts with `hqdb_AP01`.

An inbound server also uses other processes. Information about an inbound server might be recorded in the trace file for one or more of these processes. The process name of the reader server and apply servers is `AS`*`nn`*, where *`nn`* can include letters and numbers. So, on some operating systems, if the system identifier for a database running an inbound server is `hqdb` and the process number is `01`, then the trace file that contains information about a process used by an inbound server starts with `hqdb_AS01`.

See Also:

- "Displaying Session Information for Inbound Servers"
- *Oracle Database Administrator’s Guide*for more information about trace files and the alert log, and for more information about their names and locations
- *Oracle Database PL/SQL Packages and Types Reference*for more information about setting the `trace_level` apply parameter
- Your operating system specific Oracle documentation for more information about the names and locations of trace files

Problems and Solutions for XStream In

You can implement solutions for common problems with XStream In.

In general, you can troubleshoot XStream inbound servers in the same way that you troubleshoot Oracle Apply processes.

- XStream In Cannot Identify an Inbound Server
  When an XStream In configuration cannot identify an inbound server, then there might be multiple subscribers to the inbound server’s queue.
- Inbound Server Encounters an ORA-03135 Error
  If the connection is broken between the inbound server and the XStream client application, restart the client application.
- Changes Are Failing to Reach the Client Application in XStream In
  In an XStream In configuration, database changes that should be streamed to apply handlers or to the XStream client application are not reaching the apply handler or client application.

XStream In Cannot Identify an Inbound Server

When an XStream In configuration cannot identify an inbound server, then there might be multiple subscribers to the inbound server’s queue.

If an XStream In configuration cannot identify an inbound server, then an error is returned.

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 OWNER, QUEUE_NAME, CONSUMER_NAME, ADDRESS
     FROM DBA_QUEUE_SUBSCRIBERS;
   ```

   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:

"Configuring XStream In"

Inbound Server Encounters an ORA-03135 Error

If the connection is broken between the inbound server and the XStream client application, restart the client application.

An inbound server encounters the following error:

```
ORA-03135: connection lost contact
```

Problem: Connection Broken Between the Inbound Server and the Client Application

The `ORA-03135` error indicates that the connection between the inbound server and the XStream client application was broken.

Solution

To correct the problem:

- Restart the XStream client application.

See Also:

"Sample XStream Client Application"

Changes Are Failing to Reach the Client Application in XStream In

In an XStream In configuration, database changes that should be streamed to apply handlers or to the XStream client application are not reaching the apply handler or client application.

The following sections describe possible problems and their solutions.

Problem: LCRs Blocked in the Stream

LCRs might be blocked after reaching the inbound server. For example, the inbound server might be encountering errors and moving transactions to the error queue, 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` apply parameter to `1` or another relatively low value

  To disable LCR tracking when you use this method, set the `message_tracking_frequency` apply parameter to `NULL` or exit the session.

- Running the `SET_MESSAGE_TRACKING` procedure in the `DBMS_XSTREAM_ADM` package

  To disable LCR tracking when you use this method, set the `tracking_label` parameter to `NULL` in the `SET_MESSAGE_TRACKING` procedure or exit the session.

After using one of these methods, use the `V$XSTREAM_MESSAGE_TRACKING` view to monitor the progress of LCRs through a stream. If you are using Oracle GoldenGate to process the LCR, then you can use the `V$GOLDENGATE_MESSAGE_TRACKING` view to monitor the progress of LCRs through Oracle GoldenGate components. By tracking an LCR through the stream, you can determine where the LCR is blocked.

Solution

To correct problem:

- Take the appropriate action based on the reason that the LCR is blocked. For example, the following actions might correct the problem:
  - If an inbound server is encountering errors, then correct the problem that is causing the errors.
  - If an apply handler is not processing LCRs correctly, then correct the apply handler.
  - If an Oracle GoldenGate component is not processing LCRs correctly, then correct the Oracle GoldenGate component.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `message_tracking_frequency` apply parameter
- The Oracle GoldenGate documentation for more information about Oracle GoldenGate

How to Get More Help with XStream In

Oracle Support can provide more help with XStream In.

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 IV
Appendixes

Appendixes include information about XStream client applications and XStream restrictions.

- Sample XStream Client Application
  Examples illustrate how to configure the Oracle Database components that are used by XStream. The examples configure sample client applications that communicate with an XStream outbound server and inbound server.
- XStream Out Restrictions
  Restrictions apply to XStream Out.
- XStream In Restrictions
  Restrictions apply to XStream In.

A

Sample XStream Client Application

Examples illustrate how to configure the Oracle Database components that are used by XStream. The examples configure sample client applications that communicate with an XStream outbound server and inbound server.

- About the Sample XStream Client Application
  A sample XStream client application illustrates the basic tasks that are required of an XStream Out and XStream In application.
- 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.
- 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.

See Also:

- "XStream Out Concepts"
- "XStream Use Cases"
- *Oracle Call Interface Developer's Guide*
- *Oracle Database XStream Java API Reference*

About the Sample XStream Client Application

A sample XStream client application illustrates the basic tasks that are required of an XStream Out and XStream In application.

The application performs the following tasks:

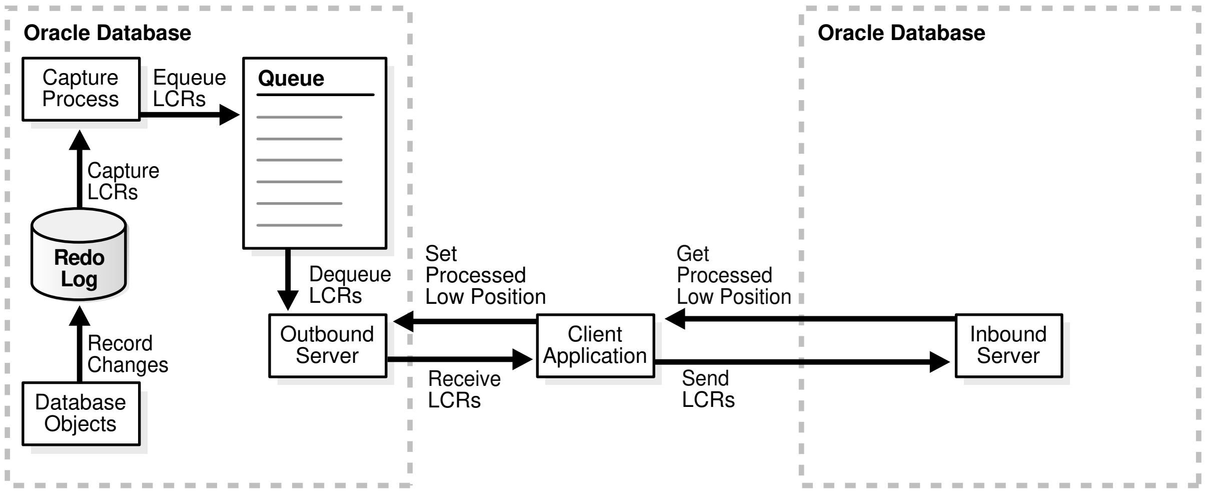
- It attaches to an XStream outbound server and inbound server and waits for LCRs from the outbound server. The outbound server and inbound server are in two different databases.
- When it receives an LCR from the outbound server, it immediately sends the LCR to the inbound server.
- It periodically 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 A-1 provides an overview of the XStream environment configured in this section.

Figure A-1 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

If you are running the sample application with a multitenant container database (CDB), then ensure that the client application connects to the correct container:

- When the client application connects to the outbound server, it must connect to the root.
- When the client application connects to the inbound server, it must connect to the container in which the inbound server was created.

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 Order in an LCR Stream"
- "Configuring XStream Out"
- "Configuring XStream In"
- *Oracle Database PL/SQL Packages and Types Reference*

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.

Next, 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((char *)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%lx\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 = ");
  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 = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ob_usr", 6))
    {
      xout_params->user = (oratext *)value;
      xout_params->userlen = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ob_pwd", 6))
    {
      xout_params->passw = (oratext *)value;
      xout_params->passwlen = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ob_svr", 6))
    {
      xout_params->svrnm = (oratext *)value;
      xout_params->svrnmlen = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_db", 5))
    {
      xin_params->dbname = (oratext *)value;
      xin_params->dbnamelen = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_usr", 6))
    {
      xin_params->user = (oratext *)value;
      xin_params->userlen = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_pwd", 6))
    {
      xin_params->passw = (oratext *)value;
      xin_params->passwlen = (ub4)strlen(value);
    }
    else if (!strncmp(option, (char *)"ib_svr", 6))
    {
      xin_params->svrnm = (oratext *)value;
      xin_params->svrnmlen = (ub4)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.

Next, 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:
0000000920250000000001000000010000000920250000000010000000101
Attached to outbound server:xout
Last Position is: 0000000920250000000001000000010000000920250000000010000000101
```

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)
{
  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);
```

```
        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();
        }
        else  // batch is end
        {
          assert alcr == null;
          // 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("");
  }
}
```

B XStream Out Restrictions

Restrictions apply to XStream Out.

- Capture Process Restrictions
  Restrictions apply to capture processes.
- Propagation Restrictions
  Restrictions apply to propagations.
- Outbound Server Restrictions
  Restrictions apply to outbound servers.
- XStream Out Rule Restrictions
  Restrictions apply to rules.
- XStream Out Rule-Based Transformation Restrictions
  Restrictions apply to rule-based transformations in XStream Out.
- XStream Out Limitations for Extended Data Types
  Some restrictions apply to extended data types in XStream Out.

Capture Process Restrictions

Restrictions apply to capture processes.

- Unsupported Data Types for Capture Processes
  Capture processes do not support some data types.
- Unsupported Changes for Capture Processes
  Capture processes do not support some changes.
- Supplemental Logging Data Type Restrictions
  Some types of columns cannot be part of a supplemental log group.
- Operational Requirements for Downstream XStream Out with XStream Out
  There are operational requirements for downstream XStream Out with XStream Out.
- Capture Processes Do Not Support Oracle Label Security
  Capture processes do not support database objects that use Oracle Label Security (OLS).

Unsupported Data Types for Capture Processes

Capture processes do not support some data types.

A capture process does not capture the results of DML changes to columns of the following data types:

- `ROWID`
- Nested tables
- The following Oracle-supplied types: `ANYTYPE`, `ANYDATASET`, URI types, `SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`, and `Expression`

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

Note:

XStream does not support `LONG` columns in databases with varying width multibyte character sets.

See Also:

"Data Types Captured by a Capture Process"

Capture processes can capture changes to SecureFiles LOB columns only if the source database compatibility level is set to 11.2.0.0 or higher. Also, capture processes do not support capturing changes resulting from fragment-based operations on SecureFiles LOB columns or capturing changes resulting from SecureFiles archive manager operations.

When a capture process tries to create a row LCR for a DML change to a column of an unsupported data type, the capture process can either ignore the change to the table or raise an error. The behavior of the capture process depends on the setting for the `ignore_unsupported_table` capture process parameter.

When the capture process ignores the change to the table, it does not capture the change, and it records the table name in the alert log. When the capture process raises an error, it writes the LCR that caused the error into its trace file, raises an ORA-26744 error, and becomes disabled. In either case, modify the rules used by the capture process to avoid recording messages in the alert log or capture process errors. After modifying the capture process's rules, restart the capture process.

Note:

- You can add rules to a negative rule set for a capture process that instruct the capture process to discard changes to tables with columns of unsupported data types.
- Capture processes do not support primary keys that contain object type attributes.
- A capture process raises an error if it attempts to capture an `INSERT` operation with an `APPEND` hint if the `INSERT` operation includes a column of either of the following types: XMLType stored as object relational or XMLType stored as binary XML

See Also:

- "Rules and Rule Sets"
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ignore_unsupported_table` capture process parameter
- *Oracle Database Utilities* for more information about LogMiner restrictions for SecureFiles LOB columns
- *Oracle Database Upgrade Guide* for information about database compatibility

Unsupported Changes for Capture Processes

Capture processes do not support some changes.

- Unsupported Schemas for Capture Processes
  Capture processes do not support some schemas.
- Unsupported Table Types for Capture Processes
  Capture processes do not support some table types.
- Unsupported DDL Changes for Capture Processes
  Capture processes do not support some data definition language (DDL) changes.
- Changes Ignored by a Capture Process
  Capture processes ignore some types of changes.
- NOLOGGING and UNRECOVERABLE Keywords for SQL Operations
  If you use the `NOLOGGING` or `UNRECOVERABLE` keyword for a SQL operation, then the changes resulting from the SQL operation cannot be captured by a capture process.
- UNRECOVERABLE Clause for Direct Path Loads
  If you use the `UNRECOVERABLE` clause in the SQL*Loader control file for a direct path load, then a capture process cannot capture the changes resulting from the direct path load.

Unsupported Schemas for Capture Processes

Capture processes do not support some schemas.

By default, a capture process does not capture changes made to the following schemas:

- `CTXSYS`
- `DBSNMP`
- `DMSYS`
- `DVSYS`
- `EXFSYS`
- `LBACSYS`
- `MDDATA`
- `MDSYS`
- `OLAPSYS`
- `ORDDATA`

- `ORDPLUGINS`
- `ORDSYS`
- `OUTLN`
- `SI_INFORMTN_SCHEMA`
- `SYS`
- `SYSMAN`
- `SYSTEM`
- `WMSYS`
- `XDB`

If the `include_objects` capture process parameter specifies one or more of these schemas, then the capture process captures changes made to the specified schemas. If the `include_objects` capture process parameter specifies one or more tables in these schemas, then the capture process captures changes made to the specified tables.

By default, the `include_objects` capture process parameter is set to `NULL`. Therefore, the capture process does not capture changes made to these schemas.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `include_objects` capture process parameter

Unsupported Table Types for Capture Processes

Capture processes do not support some table types.

A capture process cannot capture DML changes made to the following types of tables:

- Temporary tables
- Object tables that include the unsupported data types described in "Unsupported Data Types for Capture Processes"

Note:

- A capture process can capture changes to tables compressed with basic table compression and OLTP table compression only if the compatibility level at both the source database and the capture database is set to 11.2.0.0.0 or higher.
- A capture process can capture changes to tables compressed with hybrid columnar compression if all of the following conditions are met: both the source database and the capture database must be running Oracle Database 11*g* Release 2 (11.2.0.2), and the compatibility level at both the source database and the capture database is set to 11.2.0.0.0 or higher.

See Also:

- "Data Types Captured by a Capture Process"
- *Oracle Database Administrator's Guide* for information about compressed tables

Unsupported DDL Changes for Capture Processes

Capture processes do not support some data definition language (DDL) changes.

A capture process captures the DDL changes that satisfy its rule sets, *except for* the following types of DDL changes:

- `ALTER DATABASE`
- `CREATE CONTROLFILE`
- `CREATE DATABASE`
- `CREATE PFILE`
- `CREATE SPFILE`

A capture process can capture DDL statements, but not the results of DDL statements, unless the DDL statement is a `CREATE TABLE AS SELECT` statement. For example, when a capture process captures an `ANALYZE` statement, it does not capture the statistics generated by the `ANALYZE` statement. However, when a capture process captures a `CREATE TABLE AS SELECT` statement, it captures the statement itself and all of the rows selected (as `INSERT` row LCRs).

Some types of DDL changes that are captured by a capture process cannot be applied by an outbound server. If an outbound server receives a DDL LCR that specifies an operation that cannot be processed, then the outbound server ignores the DDL LCR and records information about it in its trace file.

See Also:

"Rules and Rule Sets"

Changes Ignored by a Capture Process

Capture processes ignore some types of changes.

A capture process ignores the following types of changes:

- The session control statements `ALTER SESSION` and `SET ROLE`.
- The system control statement `ALTER SYSTEM`.
- `CALL`, `EXPLAIN PLAN`, and `LOCK TABLE` statements.
- `GRANT` statements on views.
- Changes made to a table or schema by online redefinition using the `DBMS_REDEFINITION` package. Online table redefinition is supported on a table for which a capture process captures changes, but the logical structure of the table before online redefinition must be the same as the logical structure after online redefinition.

- Changes to sequence values. For example, if a user references a `NEXTVAL` or sets the sequence, then a capture process does not capture changes resulting from these operations. Also, if you share a sequence at multiple databases, then sequence values used for individual rows at these databases might vary.
- Invocations of PL/SQL procedures, which means that a call to a PL/SQL procedure is not captured. However, if a call to a PL/SQL procedure causes changes to database objects, then these changes can be captured by a capture process if the changes satisfy the capture process rule sets.

Note:

- If an Oracle-supplied package related to XML makes changes to database objects, then these changes are not captured by capture processes.
- If an Oracle-supplied package related to Oracle Text makes changes to database objects, then these changes are not captured by capture processes.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about packages related to XML
- *Oracle Text Reference* for information about packages related to Oracle Text

NOLOGGING and UNRECOVERABLE Keywords for SQL Operations

If you use the `NOLOGGING` or `UNRECOVERABLE` keyword for a SQL operation, then the changes resulting from the SQL operation cannot be captured by a capture process.

Therefore, do not use these keywords to capture the changes that result from a SQL operation.

If the object for which you are specifying the logging attributes resides in a database or tablespace in `FORCE LOGGING` mode, then Oracle Database ignores any `NOLOGGING` or `UNRECOVERABLE` setting until the database or tablespace is taken out of `FORCE LOGGING` mode. You can determine the current logging mode for a database by querying the `FORCE_LOGGING` column in the `V$DATABASE` dynamic performance view. You can determine the current logging mode for a tablespace by querying the `FORCE_LOGGING` column in the `ALL_TABLESPACES` static data dictionary view.

Note:

The `UNRECOVERABLE` keyword is deprecated and has been replaced with the `NOLOGGING` keyword in the *`logging_clause`*. Although `UNRECOVERABLE` is supported for backward compatibility, Oracle strongly recommends that you use the `NOLOGGING` keyword, when appropriate.

See Also:

*Oracle Database SQL Language Reference*for more information about the NOLOGGING and UNRECOVERABLE keywords, FORCE LOGGING mode, and the *logging_clause*

UNRECOVERABLE Clause for Direct Path Loads

If you use the UNRECOVERABLE clause in the SQL*Loader control file for a direct path load, then a capture process cannot capture the changes resulting from the direct path load.

Therefore, if the changes resulting from a direct path load should be captured by a capture process, then do not use the UNRECOVERABLE clause.

If you load objects into a database or tablespace that is in FORCE LOGGING mode, then Oracle Database ignores any UNRECOVERABLE clause during a direct path load, and the loaded changes are logged. You can determine the current logging mode for a database by querying the FORCE_LOGGING column in the V$DATABASE dynamic performance view. You can determine the current logging mode for a tablespace by querying the FORCE_LOGGING column in the DBA_TABLESPACES static data dictionary view.

See Also:

Oracle Database Utilities for information about direct path loads and SQL*Loader

Supplemental Logging Data Type Restrictions

Some types of columns cannot be part of a supplemental log group.

Columns of the following data types cannot be part of a supplemental log group: LOB, LONG, LONG RAW, user-defined types (including object types, REFs, varrays, nested tables), and Oracle-supplied types (including Any types, XML types, spatial types, and media types).

See Also:

- "If Required, Configure Supplemental Logging"
- *Oracle Database SQL Language Reference* for information about data types

Operational Requirements for Downstream XStream Out with XStream Out

There are operational requirements for downstream XStream Out with XStream Out.

The following are operational requirements for using downstream XStream Out:

- The source database must be running at least Oracle Database 12*c* Release 2 (12.2).

- The operating system on the source and downstream XStream Out sites must be the same, but the operating system release does not need to be the same. In addition, the downstream sites can use a different directory structure than the source site.
- The hardware architecture on the source and downstream XStream Out sites must be the same. For example, a downstream XStream Out configuration with a source database on a 64-bit Sun system must have a downstream database that is configured on a 64-bit Sun system. Other hardware elements, such as the number of CPUs, memory size, and storage configuration, can be different between the source and downstream sites.

See Also:

"Local Capture and Downstream Capture"

Capture Processes Do Not Support Oracle Label Security

Capture processes do not support database objects that use Oracle Label Security (OLS).

See Also:

Oracle Label Security Administrator's Guide

Propagation Restrictions

Restrictions apply to propagations.

- Connection Qualifiers and Propagations
  Connection qualifiers cannot be specified in the database links that are used by propagations.

Connection Qualifiers and Propagations

Connection qualifiers cannot be specified in the database links that are used by propagations.

Outbound Server Restrictions

Restrictions apply to outbound servers.

- Unsupported Data Types for Outbound Servers
  Outbound servers do not support some data types.
- Types of DDL Changes Ignored by an Outbound Server
  Outbound servers do not support some types of DDL changes.
- Apply Process Features That Are Not Applicable to Outbound Servers
  Some features cannot be used with outbound servers.

Unsupported Data Types for Outbound Servers

Outbound servers do not support some data types.

An outbound server does not process row LCRs containing the results of DML changes in columns of the following data types:

- `ROWID`
- Nested tables
- The following Oracle-supplied types: `ANYTYPE`, `ANYDATASET`, URI types, `SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`, and `Expression`

Note:

XStream does not support `LONG` columns in databases with varying width multibyte character sets.

An outbound server raises an error if it attempts to process a row LCR that contains information about a column of an unsupported data type. In addition, an outbound server cannot process DML changes to the following types of tables:

- Temporary tables
- Object tables that include unsupported data types

An outbound server raises an error if it attempts to process such changes. When an outbound server raises an error for an LCR, it moves the transaction that includes the LCR into the error queue.

See Also:

- "Data Types Supported by Outbound Servers"
- *Oracle Database SQL Language Reference* for information about data types

Types of DDL Changes Ignored by an Outbound Server

Outbound servers do not support some types of DDL changes.

The following types of DDL changes are not supported by an outbound server:

- `CREATE DATABASE LINK`
- `CREATE SCHEMA AUTHORIZATION`
- `DROP DATABASE LINK`
- `FLASHBACK DATABASE`
- `RENAME`

XStream OUT processes DDL as a LCR$_DDL_RECORD in which the DDL text is a string. It is the responsibility of the target application or database how this will be handled. Not all DDL generated by XStream OUT can be applied by XStream IN.

If an outbound server receives a DDL LCR that specifies an operation that cannot be processed, then the outbound server ignores the DDL LCR and records the following message in the outbound server trace file, followed by the DDL text that was ignored:

```
Apply process ignored the following DDL:
```

An outbound server applies all other types of DDL changes if the DDL LCRs containing the changes should be applied according to the outbound server rule sets.

> **Note:**
>
> - Instead of using the command `RENAME TABLE`, you can use the `ALTER TABLE jobs RENAME` command which is supported and is an alternative solution.
> - The name "materialized view" is synonymous with the name "snapshot". Snapshot equivalents of the statements on materialized views are ignored by an outbound server.

Related Topics

- Types of DDL Changes Ignored by an Inbound Server
  Inbound servers ignore some types of DDL changes.

> **See Also:**
>
> Rules and Rule Sets

Apply Process Features That Are Not Applicable to Outbound Servers

Some features cannot be used with 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.
- The following apply 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 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.

See Also:

Oracle Database PL/SQL Packages and Types Reference

- Apply tags

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

- Apply database links

  Outbound servers cannot use database links.

- Conflict detection and resolution

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

- Dependency scheduling

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

- Substitute key column settings

  An outbound server ignores substitute key column settings.

- 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 Also:

  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 Also:

  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` view, but it does not enqueue the transaction into an error queue.

XStream Out Rule Restrictions

Restrictions apply to rules.

- Restrictions for Subset Rules
  Restrictions apply to subset rules.

Restrictions for Subset Rules

Restrictions apply to subset rules.

The following restrictions apply to subset rules:

- A table with the table name referenced in the subset rule must exist in the same database as the subset rule, and this table must be in the same schema referenced for the table in the subset rule.
- If the subset rule is in the positive rule set for a capture process, then the table must contain the columns specified in the subset condition, and the data type of each of these columns must match the data type of the corresponding column at the source database.
- If the subset rule is in the positive rule set for a propagation, then the table must contain the columns specified in the subset condition, and the data type of each column must match the data type of the corresponding column in row LCRs that evaluate to `TRUE` for the subset rule.
- Creating subset rules for tables that have one or more columns of the following data types is not supported: LOB, `LONG`, `LONG RAW`, nested tables, and Oracle-supplied types (including `Any` types, XML types, spatial types, and media types).

See Also:

- "Subset Rules"
- *Oracle Database SQL Language Reference* for more information about data types

XStream Out Rule-Based Transformation Restrictions

Restrictions apply to rule-based transformations in XStream Out.

- Unsupported Data Types for Declarative Rule-Based Transformations
  Except for add column transformations, declarative rule-based transformations that operate on columns support the same data types that are supported by capture processes.

See Also:

"Rule-Based Transformations"

Unsupported Data Types for Declarative Rule-Based Transformations

Except for add column transformations, declarative rule-based transformations that operate on columns support the same data types that are supported by capture processes.

Add column transformations cannot add columns of the following data types: `BLOB`, `CLOB`, `NCLOB`, `BFILE`, `LONG`, `LONG RAW`, `ROWID`, nested tables, and Oracle-supplied types (including `Any` types, XML types, spatial types, and media types).

Extended data type columns cannot be used in the following types of declarative rule-based transformations:

- Add column
- Keep columns

See Also:

- "Data Types Captured by a Capture Process"
- "Unsupported Data Types for Capture Processes"
- "XStream Out Limitations for Extended Data Types"
- *Oracle Database SQL Language Reference* for information about data types

XStream Out Limitations for Extended Data Types

Some restrictions apply to extended data types in XStream Out.

The maximum size of the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types has been increased in Oracle Database 12*c* when the `COMPATIBLE` initialization parameter is set to `12.0.0` and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`. XStream Out supports these extended data types.

However, the following limitations apply to the extended data type:

- Information about an extended data type column might not be contained in the original LCR for a data manipulation language (DML) operation. Instead, XStream Out might treat the extended data type column similar to the way it treats LOB columns. Specifically, additional LCRs might contain the information for the extended data type column.
- XStream rules cannot access data in LCRs for extended data type columns.
- Extended data type columns cannot be specified in a subset rule clause.
- Extended data type columns cannot be used in the following types of declarative rule-based transformations:
  - Add column
  - Keep columns

See Also:

Oracle Database SQL Language Reference for more information about extended data types

C
XStream In Restrictions

Restrictions apply to XStream In.

- Inbound Server Restrictions
  Restrictions apply to inbound servers.
- XStream In Rule Restrictions
  Restrictions apply to rules.
- XStream In Rule-Based Transformation Restrictions
  Restrictions apply to rule-based transformations in XStream In.
- XStream In Limitations for Extended Data Types
  Limitations apply to extended data types in XStream In.

Inbound Server Restrictions

Restrictions apply to inbound servers.

- Unsupported Data Types for Inbound Servers
  Inbound servers do not support some data types.
- Unsupported Data Types for Apply Handlers
  Apply handlers do not support some data types.
- Types of DDL Changes Ignored by an Inbound Server
  Inbound servers ignore some types of DDL changes.
- Current Schema User Must Exist at Destination Database
  For a DDL LCR to be applied at a destination database successfully, the user specified as the `current_schema` in the DDL LCR must exist at the destination database.
- Inbound Servers Do Not Support Oracle Label Security
  Inbound servers do not support database objects that use Oracle Label Security (OLS).

Unsupported Data Types for Inbound Servers

Inbound servers do not support some data types.

An inbound server does not apply row LCRs containing the results of DML changes in columns of the following data types:

- `ROWID`
- Nested tables
- The following Oracle-supplied types: `ANYTYPE`, `ANYDATASET`, URI types, `SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`, and `Expression`

Note:

XStream does not support `LONG` columns in databases with varying width multibyte character sets.

An inbound server raises an error if it attempts to apply a row LCR that contains information about a column of an unsupported data type. In addition, an inbound server cannot apply DML changes to the following types of tables:

- Temporary tables
- Object tables that include unsupported data types

An inbound server raises an error if it attempts to apply such changes. When an inbound server raises an error for an LCR, it moves the transaction that includes the LCR into the error queue.

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

See Also:

- "Data Types Applied by Inbound Servers"
- *Oracle Database SQL Language Reference* for information about data types

Unsupported Data Types for Apply Handlers

Apply handlers do not support some data types.

Procedure DML handlers and error handlers cannot process `LONG` or `LONG RAW` column data in row LCRs. However, procedure DML handlers and error handlers can process both nonassembled and assembled LOB column data in row LCRs, but these handlers cannot modify nonassembled LOB column data.

See Also:

- "LCR Processing Options for Inbound Servers"
- *Oracle Database SQL Language Reference* for more information about data types

Types of DDL Changes Ignored by an Inbound Server

Inbound servers ignore some types of DDL changes.

The following types of DDL changes are not supported by an inbound server. These types of DDL changes are not applied:

- `ALTER MATERIALIZED VIEW`
- `ALTER MATERIALIZED VIEW LOG`
- `CREATE DATABASE LINK`
- `CREATE SCHEMA AUTHORIZATION`
- `CREATE MATERIALIZED VIEW`
- `CREATE MATERIALIZED VIEW LOG`
- `DROP DATABASE LINK`
- `DROP MATERIALIZED VIEW`
- `DROP MATERIALIZED VIEW LOG`
- `FLASHBACK DATABASE`
- `RENAME`

If an inbound server receives a DDL LCR that specifies an operation that cannot be applied, then the inbound server ignores the DDL LCR and records the following message in the inbound server trace file, followed by the DDL text that was ignored:

```
Inbound server ignored the following DDL:
```

An inbound server applies all other types of DDL changes if the DDL LCRs containing the changes should be applied according to the inbound server rule sets.

Note:

- An inbound server applies `ALTER` *`object_type object_name`* `RENAME` changes, such as `ALTER TABLE jobs RENAME`. Therefore, if you want DDL changes that rename objects to be applied, then use `ALTER` *`object_type object_name`* `RENAME` statements instead of `RENAME` statements. After changing the name of a database object, new rules that specify the new database object name might be needed to replicate changes to the database object.
- The name "materialized view" is synonymous with the name "snapshot". Snapshot equivalents of the statements on materialized views are ignored by an inbound server.

See Also:

"Rules and Rule Sets"

Current Schema User Must Exist at Destination Database

For a DDL LCR to be applied at a destination database successfully, the user specified as the `current_schema` in the DDL LCR must exist at the destination database.

The current schema is the schema that is used if no schema is specified for an object in the DDL text.

See Also:

- *Oracle Database Concepts*for more information about database structures
- *Oracle Database PL/SQL Packages and Types Reference*for more information about the `current_schema` attribute in DDL LCRs

Inbound Servers Do Not Support Oracle Label Security

Inbound servers do not support database objects that use Oracle Label Security (OLS).

See Also:

Oracle Label Security Administrator’s Guide

XStream In Rule Restrictions

Restrictions apply to rules.

- Restrictions for Subset Rules
  Restrictions apply to subset rules.

Restrictions for Subset Rules

Restrictions apply to subset rules.

The following restrictions apply to subset rules:

- A table with the table name referenced in the subset rule must exist in the same database as the subset rule, and this table must be in the same schema referenced for the table in the subset rule.
- If the subset rule is in the positive rule set for an inbound server, then the table must contain the columns specified in the subset condition, and the data type of each column must match the data type of the corresponding column in row LCRs that evaluate to `TRUE` for the subset rule.
- Creating subset rules for tables that have one or more columns of the following data types is not supported: LOB, `LONG`, `LONG RAW`, user-defined types (including object types, `REF`s, varrays, nested tables), and Oracle-supplied types (including `Any` types, XML types, spatial types, and media types).

See Also:

- "Subset Rules"
- *Oracle Database SQL Language Reference* for more information about data types

XStream In Rule-Based Transformation Restrictions

Restrictions apply to rule-based transformations in XStream In.

- Unsupported Data Types for Declarative Rule-Based Transformations
  Except for add column transformations, declarative rule-based transformations that operate on columns support the same data types that are supported by inbound servers.

See Also:

"Rule-Based Transformations"

Unsupported Data Types for Declarative Rule-Based Transformations

Except for add column transformations, declarative rule-based transformations that operate on columns support the same data types that are supported by inbound servers.

Add column transformations cannot add columns of the following data types: `BLOB`, `CLOB`, `NCLOB`, `BFILE`, `LONG`, `LONG RAW`, `ROWID`, user-defined types (including object types, `REF`s, varrays, nested tables), and Oracle-supplied types (including `Any` types, XML types, spatial types, and media types).

See Also:

- "Data Types Applied by Inbound Servers"
- "Unsupported Data Types for Inbound Servers"
- *Oracle Database SQL Language Reference* for information about data types

XStream In Limitations for Extended Data Types

Limitations apply to extended data types in XStream In.

The maximum size of the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types has been increased in Oracle Database 12*c* when the `COMPATIBLE` initialization parameter is set to `12.0.0` and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`. XStream In supports these extended data types.

However, the following limitations apply to the extended data types:

- Information about an extended data type column might not be contained in the original LCR for a data manipulation language (DML) operation. Instead, XStream In might treat the extended data type column similar to the way it treats LOB columns. Specifically, additional LCRs might contain the information for the extended data type column.
- XStream rules cannot access data in LCRs for extended data type columns.
- Extended data type columns cannot be specified in a subset rule clause.
- Extended data type columns cannot be used for conflict detection.
- Extended data type columns cannot be used for a substitute primary key for apply purposes with the `DBMS_APPLY_ADM.SET_KEY_COLUMNS` procedure.
- Extended data type columns cannot be used in the following types of declarative rule-based transformations:
  - Add column
  - Keep columns

See Also:

Oracle Database SQL Language Reference for more information about extended data types