25 DBMS_APPLY_ADM

The `DBMS_APPLY_ADM` package provides subprograms to configure and manage Oracle Apply processes, XStream outbound servers, and XStream inbound servers.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. 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.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Summary of DBMS_APPLY_ADM Subprograms

See Also:

Oracle Database XStream Guide for more information about XStream outbound servers and inbound servers

DBMS_APPLY_ADM Overview

The `DBMS_APPLY_ADM` package provides interfaces to start, stop, and configure Oracle Apply processes, XStream outbound servers, and XStream inbound servers.

This package includes subprograms for configuring apply handlers, setting enqueue destinations for messages, and specifying execution directives for messages. This package also provides administrative subprograms that set the instantiation SCN for objects at a destination database. This package also includes subprograms for managing apply errors.

XStream inbound servers and outbound servers can be used in an XStream configuration in a multitenant container database (CDB). A CDB is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs).

Note:

- For simplicity, this chapter refers to apply processes, XStream outbound servers, and XStream inbound servers as **apply components**. This chapter identifies a specific type of apply component when necessary.
- Using XStream requires purchasing a license for the Oracle GoldenGate product.

See Also:

- *Oracle Database XStream Guide*
- *Oracle Database Concepts* for more information about CDBs and PDBs

DBMS_APPLY_ADM Security Model

Security on this package can be controlled by either granting `EXECUTE` on this package to selected users or roles, or by granting `EXECUTE_CATALOG_ROLE` to selected users or roles..

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

When the `DBMS_APPLY_ADM` package is used to manage an Oracle Replication configuration, it requires that the user is granted the privileges of an Oracle Replication administrator.

When the `DBMS_APPLY_ADM` package is used to manage an XStream configuration, it requires that the user is granted the privileges of an XStream administrator.

Note:

The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as setting an apply user. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.

See Also:

Oracle Database XStream Guide for information about configuring an XStream administrator

DBMS_APPLY_ADM Deprecated Subprograms

The `NONE` value for the `commit_serialization` apply component parameter is deprecated. It is replaced by the `DEPENDENT_TRANSACTIONS` value.

Note:

Oracle recommends that you do not use deprecated apply component parameter values. Support for deprecated features is for backward compatibility only.

See Also:

SET_PARAMETER Procedure

Summary of DBMS_APPLY_ADM Subprograms

This table topic lists and describes the `DBMS_APPLY_ADM` subprograms.

Table 25-1 DBMS_APPLY_ADM Package Subprograms

Subprogram	Description
ALTER_APPLY Procedure	Alters an apply component
CLEAR_KEY_COLUMNS Procedure	Removes the key columns that were used as the substitute primary key by the `SET_KEY_COLUMNS` procedure
COMPARE_OLD_VALUES Procedure	Specifies whether to compare the old value of one or more columns in a row logical change record (row LCR) with the current value of the corresponding columns at the destination site during apply
CREATE_APPLY Procedure	Creates an apply component
CREATE_OBJECT_DEPENDENCY Procedure	Creates an object dependency
DELETE_ALL_ERRORS Procedure	Deletes all the error transactions for the specified apply component
DELETE_ERROR Procedure	Deletes the specified error transaction
DROP_APPLY Procedure	Drops an apply component
DROP_OBJECT_DEPENDENCY Procedure	Drops an object dependency
EXECUTE_ALL_ERRORS Procedure	Reexecutes the error transactions for the specified apply component
EXECUTE_ERROR Procedure	Reexecutes the specified error transaction
GET_ERROR_MESSAGE Function	Returns the message payload from the error queue for the specified message number and transaction identifier
HANDLE_COLLISIONS	Enables or disables basic conflict resolution for an apply process and a table

Table 25-1 (Cont.) DBMS_APPLY_ADM Package Subprograms

Subprogram	Description
SET_DML_CONFLICT_HANDLER Procedure	Adds, modifies, or removes a prebuilt DML conflict handler for `INSERT`, `UPDATE`, or `DELETE` conflicts on the specified object
SET_DML_HANDLER Procedure	Sets a user procedure as a procedure DML handler for a specified operation on a specified database object for a single apply component or for all apply components in the database
SET_ENQUEUE_DESTINATION Procedure	Sets the queue where the apply component automatically enqueues a message that satisfies the specified rule
SET_EXECUTE Procedure	Specifies whether a message that satisfies the specified rule is executed by an apply component
SET_GLOBAL_INSTANTIATION_SCN Procedure	Records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas
SET_KEY_COLUMNS Procedures	Records the set of columns to be used as the substitute primary key for local apply purposes and removes existing substitute primary key columns for the specified object if they exist
SET_PARAMETER Procedure	Sets an apply parameter to the specified value
SET_REPERROR_HANDLER Procedure	Specifies how a particular error is handled based on its error number
SET_SCHEMA_INSTANTIATION_SCN Procedure	Records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database
SET_TABLE_INSTANTIATION_SCN Procedure	Records the specified instantiation SCN for the specified table in the specified source database
SET_UPDATE_CONFLICT_HANDLER Procedure	Adds, updates, or drops an update conflict handler for the specified object
SET_VALUE_DEPENDENCY Procedure	Sets or removes a value dependency
START_APPLY Procedure	Directs the apply component to start applying messages
STOP_APPLY Procedure	Stops the apply component from applying any messages and rolls back any unfinished transactions being applied

Note:

All procedures commit unless specified otherwise. However, the `GET_ERROR_MESSAGE` function does not commit.

ALTER_APPLY Procedure

This procedure alters an apply component.

Syntax

```
DBMS_APPLY_ADM.ALTER_APPLY(
   apply_name                 IN  VARCHAR2,
   rule_set_name              IN  VARCHAR2  DEFAULT NULL,
   remove_rule_set            IN  BOOLEAN   DEFAULT FALSE,
   message_handler            IN  VARCHAR2  DEFAULT NULL
   remove_message_handler     IN  BOOLEAN   DEFAULT FALSE,
   ddl_handler                IN  VARCHAR2  DEFAULT NULL,
   remove_ddl_handler         IN  BOOLEAN   DEFAULT FALSE,
   apply_user                 IN  VARCHAR2  DEFAULT NULL,
   apply_tag                  IN  RAW       DEFAULT NULL,
   remove_apply_tag           IN  BOOLEAN   DEFAULT FALSE,
   precommit_handler          IN  VARCHAR2  DEFAULT NULL,
   remove_precommit_handler   IN  BOOLEAN   DEFAULT FALSE,
   negative_rule_set_name     IN  VARCHAR2  DEFAULT NULL,
   remove_negative_rule_set   IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 25-2 ALTER_APPLY Procedure Parameters

Parameter	Description
`apply_name`	The name of the apply component being altered. You must specify the name of an existing apply component. Do not specify an owner.
`rule_set_name`	The name of the positive rule set for the apply component. The positive rule set contains the rules that instruct the apply component to apply messages. If you want to use a positive rule set for the apply component, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a positive rule set in the `hr` schema named `job_apply_rules`, enter `hr.job_apply_rules`. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. If you specify `NULL` and the `remove_rule_set` parameter is set to `FALSE`, then this procedure retains any existing positive rule set for the specified apply component. If you specify `NULL` and the `remove_rule_set` parameter is set to `TRUE`, then this procedure removes any existing positive rule set from the specified apply component.

Table 25-2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
`remove_rule_set`	If `TRUE`, then the procedure removes the positive rule set for the specified apply component. If you remove the positive rule set for an apply component, and the apply component does not have a negative rule set, then the apply component dequeues all messages in its queue. If you remove the positive rule set for an apply component, and a negative rule set exists for the apply component, then the apply component dequeues all messages in its queue that are not discarded by the negative rule set. If `FALSE`, then the procedure retains the positive rule set for the specified apply component. If the `rule_set_name` parameter is non-`NULL`, then this parameter should be set to `FALSE`.
`message_handler`	A user-defined procedure that processes non-LCR messages in the queue for the apply component. See "Usage Notes" in the CREATE_APPLY Procedure for more information about a message handler procedure.
`remove_message_handler`	If `TRUE`, then the procedure removes the message handler for the specified apply component. If `FALSE`, then the procedure retains any message handler for the specified apply component. If the `message_handler` parameter is non-`NULL`, then this parameter should be set to `FALSE`.
`ddl_handler`	A user-defined procedure that processes DDL logical change records (DDL LCRs) in the queue for the apply component. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the `EXECUTE` member procedure of a DDL LCR, then a commit is performed automatically. See "Usage Notes" in the CREATE_APPLY Procedure for more information about a DDL handler procedure.
`remove_ddl_handler`	If `TRUE`, then the procedure removes the DDL handler for the specified apply component. If `FALSE`, then the procedure retains any DDL handler for the specified apply component. If the `ddl_handler` parameter is non-`NULL`, then this parameter should be set to `FALSE`.

Table 25-2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
`apply_user`	The user in whose security domain an apply component dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations, and runs apply handlers. If `NULL`, then the apply user is not changed. If a non-`NULL` value is specified to change the apply user, then the user who invokes the `ALTER_APPLY` procedure must be granted the `DBA` role. Only the `SYS` user can set the `apply_user` to `SYS`. If you change the apply user, then this procedure grants the new apply user dequeue privilege on the queue used by the apply component. It also configures the user as a secure queue user of the queue. In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user: • The necessary privileges to perform DML and DDL changes on the apply objects • `EXECUTE` privilege on the rule sets used by the apply component • `EXECUTE` privilege on all rule-based transformation functions used in the rule set • `EXECUTE` privilege on all apply handler procedures These privileges can be granted directly to the apply user, or they can be granted through roles. In addition, the apply user must be granted the `EXECUTE` privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply component. These privileges must be granted directly to the apply user. They cannot be granted through roles. By default, this parameter is set to the user who created the apply component by running either the `CREATE_APPLY` procedure in this package. **Note:** If the apply user for an apply component is dropped using `DROP USER. . . CASCADE`, then the apply component is also dropped automatically.
`apply_tag`	A binary tag that is added to redo entries generated by the specified apply component. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply component is running captures changes made by the apply component. If so, then the captured changes include the tag specified by this parameter. If `NULL`, the default, then the apply tag for the apply component is not changed. The following is an example of a tag with a hexadecimal value of `17`: `HEXTORAW('17')`

Table 25-2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
`remove_apply_tag`	If `TRUE`, then the procedure sets the apply tag for the specified apply component to `NULL`, and the apply component generates redo entries with `NULL` tags. If `FALSE`, then the procedure retains any apply tag for the specified apply component. If the `apply_tag` parameter is non-`NULL`, then this parameter should be set to `FALSE`.
`precommit_handler`	A user-defined procedure that can receive internal commit directives in the queue for the apply component before they are processed by the apply component. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply component. An internal commit directive is enqueued in the following ways: • When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs. • When a user or application enqueues messages and then issues a `COMMIT` statement, the commit directive is enqueued automatically. For a captured row LCR, a commit directive contains the commit SCN of the transaction from the source database. For a user message, the commit SCN is generated by the apply component. The precommit handler procedure must conform to the following restrictions: • Any work that commits must be an autonomous transaction. • Any rollback must be to a named savepoint created in the procedure. If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue. See "Usage Notes" in the CREATE_APPLY Procedure for more information about a precommit handler procedure.
`remove_precommit_handler`	If `TRUE`, then the procedure removes the precommit handler for the specified apply component. If `FALSE`, then the procedure retains any precommit handler for the specified apply component. If the `precommit_handler` parameter is non-`NULL`, then this parameter should be set to `FALSE`.

Table 25-2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
`negative_rule_set_name`	The name of the negative rule set for the apply component. The negative rule set contains the rules that instruct the apply component to discard messages. If you want to use a negative rule set for the apply component, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a negative rule set in the `hr` schema named `neg_apply_rules`, enter `hr.neg_apply_rules`. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. If you specify `NULL` and the `remove_negative_rule_set` parameter is set to `FALSE`, then the procedure retains any existing negative rule set. If you specify `NULL` and the `remove_negative_rule_set` parameter is set to `TRUE`, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for an apply component, then the negative rule set is always evaluated first.
`remove_negative_rule_set`	If `TRUE`, then the procedure removes the negative rule set for the specified apply component. If you remove the negative rule set for an apply component, and the apply component does not have a positive rule set, then the apply component dequeues all messages in its queue. If you remove the negative rule set for an apply component, and a positive rule set exists for the apply component, then the apply component dequeues all messages in its queue that are not discarded by the positive rule set. If `FALSE`, then the procedure retains the negative rule set for the specified apply component. If the `negative_rule_set_name` parameter is non-`NULL`, then this parameter should be set to `FALSE`.

Usage Notes

The following usage notes apply to this procedure:

- Automatic Restart of Apply Components
- The ALTER_APPLY Procedure and XStream Outbound Servers
- The ALTER_APPLY Procedure and XStream Inbound Servers

Automatic Restart of Apply Components

An apply component is stopped and restarted automatically when you change the value of one or more of the following `ALTER_APPLY` procedure parameters:

- `message_handler`
- `ddl_handler`
- `apply_user`
- `apply_tag`
- `precommit_handler`

The ALTER_APPLY Procedure and XStream Outbound Servers

The following usage notes apply to this procedure and XStream outbound servers:

- The `apply_user` parameter can change the connect user for an outbound server.
- You cannot specify an apply handler for an outbound server. An outbound server ignores the settings for the following parameters: `message_handler`, `ddl_handler`, and `precommit_handler`.

  The client application can perform custom processing of the LCRs instead if necessary.
- An outbound server cannot set an apply tag for the changes it processes. An outbound server ignores the setting for the `apply_tag` parameter.

The ALTER_APPLY Procedure and XStream Inbound Servers

Inbound servers can use apply handlers and process only DML and DDL LCRs. Therefore, inbound servers ignore message handlers specified in the `message_handler` parameter.

CLEAR_KEY_COLUMNS Procedure

This procedure removes the key columns that were used as the substitute primary key by the `SET_KEY_COLUMNS` procedure.

Syntax

```
DBMS_APPLY_ADM.CLEAR_KEY_COLUMNS(
   apply_name   IN   VARCHAR2   DEFAULT NULL);
```

Parameter

Table 25-3 CLEAR_KEY_COLUMNS Procedure Parameter

Parameter	Description
`apply_name`	The apply component name.

Usage Notes

The following usage notes apply to this procedure:

The CLEAR_KEY_COLUMNS Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The CLEAR_KEY_COLUMNS Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

The CLEAR_KEY_COLUMNS Procedure and CDBs

This procedure removes the columns that are used as a substitute primary key. You must perform the `CLEAR_KEY_COLUMNS` procedure in the appropriate PDB.

COMPARE_OLD_VALUES Procedure

This procedure specifies whether to compare the old values of one or more columns in a row logical change record (row LCR) with the current values of the corresponding columns at the destination site during apply.

This procedure is relevant only for `UPDATE` and `DELETE` operations because only these operations result in old column values in row LCRs. The default is to compare old values for all columns.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Syntax

```
DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
   object_name          IN VARCHAR2,
   column_list          IN VARCHAR2,
   operation            IN VARCHAR2 DEFAULT 'UPDATE',
   compare              IN BOOLEAN  DEFAULT TRUE,
   apply_database_link  IN VARCHAR2 DEFAULT NULL);

DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
   object_name          IN VARCHAR2,
   column_table         IN DBMS_UTILITY.LNAME_ARRAY,
   operation            IN VARCHAR2 DEFAULT 'UPDATE',
   compare              IN BOOLEAN  DEFAULT TRUE,
   apply_database_link  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25-4 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
`object_name`	The name of the source table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`column_list`	A comma-delimited list of column names in the table. There must be no spaces between entries. Specify `*` to include all nonkey columns.
`column_table`	A PL/SQL associative array of type `DBMS_UTILITY.LNAME_ARRAY` that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be `NULL` terminated.
`operation`	The name of the operation, which can be specified as: • `UPDATE` for `UPDATE` operations • `DELETE` for `DELETE` operations • `*` for both `UPDATE` and `DELETE` operations
`compare`	If `compare` is `TRUE`, the old values of the specified columns are compared during apply. If `compare` is `FALSE`, the old values of the specified columns are not compared during apply.

Table 25-4 (Cont.) COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

Usage Notes

The following usage notes apply to this procedure:

- Conflict Detection
- The COMPARE_OLD_VALUES Procedure and XStream Outbound Servers
- The COMPARE_OLD_VALUES Procedure and XStream Inbound Servers

Conflict Detection

By default, an apply component uses the old column values in a row LCR to detect conflicts. You can choose not to compare old column values to avoid conflict detection for specific tables. For example, if you do not want to compare the old values for a set of columns during apply, then, using the `COMPARE_OLD_VALUES` procedure, specify the set of columns in the `column_list` or `column_table` parameter, and set the `compare` parameter to `FALSE`.

In addition, when the `compare_key_only` apply component parameter is set to `Y`, automatic conflict detection is disabled, and the apply component only uses primary key and unique key columns to identify the table row for a row LCR. When the `compare_key_only` apply component parameter is set to `N`, automatic conflict detection is enabled, and the apply component uses all of the old values in a row LCR to identify the table row for a row LCR.

Note:

- An apply component compares old values for non-key columns when they are present in a row LCR and when the apply component parameter `compare_key_only` is set to `N`.
- This procedure raises an error if a key column is specified in `column_list` or `column_table` and the `compare` parameter is set to `FALSE`.

See Also:

SET_PARAMETER Procedure for more information about the `compare_key_only` apply component parameter

The COMPARE_OLD_VALUES Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The COMPARE_OLD_VALUES Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

CREATE_APPLY Procedure

This procedure creates an apply component.

Syntax

```
DBMS_APPLY_ADM.CREATE_APPLY(
   queue_name               IN  VARCHAR2,
   apply_name               IN  VARCHAR2,
   rule_set_name            IN  VARCHAR2  DEFAULT NULL,
   message_handler          IN  VARCHAR2  DEFAULT NULL,
   ddl_handler              IN  VARCHAR2  DEFAULT NULL,
   apply_user               IN  VARCHAR2  DEFAULT NULL,
   apply_database_link      IN  VARCHAR2  DEFAULT NULL,
   apply_tag                IN  RAW       DEFAULT '00',
   apply_captured           IN  BOOLEAN   DEFAULT FALSE,
   precommit_handler        IN  VARCHAR2  DEFAULT NULL,
   negative_rule_set_name   IN  VARCHAR2  DEFAULT NULL,
   source_database          IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-5 CREATE_APPLY Procedure Parameters

Parameter	Description
`queue_name`	The name of the queue from which the apply component dequeues messages. You must specify an existing queue in the form `[`*`schema_name`*`.]`*`queue_name`*. For example, to specify a queue in the `hr` schema named `streams_queue`, enter `hr.streams_queue`. If the schema is not specified, then the current user is the default. **Note:** The `queue_name` setting cannot be altered after the apply component is created.
`apply_name`	The name of the apply component being created. A `NULL` specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing apply component or messaging client. **Note:** The `apply_name` setting cannot be altered after the apply component is created.
`rule_set_name`	The name of the positive rule set for the apply component. The positive rule set contains the rules that instruct the apply component to apply messages. If you want to use a positive rule set for the apply component, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a positive rule set in the `hr` schema named `job_apply_rules`, enter `hr.job_apply_rules`. If the schema is not specified, then the current user is the default. If you specify `NULL`, and no negative rule set is specified, then the apply component applies either all captured messages or all messages in the persistent queue, depending on the setting of the `apply_captured` parameter. An error is returned if the specified rule set does not exist.

Table 25-5 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
`message_handler`	A user-defined procedure that processes non-LCR messages in the queue for the apply component. See "Usage Notes" for more information about a message handler procedure.
`ddl_handler`	A user-defined procedure that processes DDL logical change record (DDL LCRs) in the queue for the apply component. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the `EXECUTE` member procedure of a DDL LCR, then a commit is performed automatically. See "Usage Notes" for more information about a DDL handler procedure.
`apply_user`	The user who applies all DML and DDL changes that satisfy the apply component rule sets and who runs user-defined apply handlers. If `NULL`, then the user who runs the `CREATE_APPLY` procedure is used. The apply user is the user in whose security domain an apply component dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations configured for apply component rules, and runs apply handlers configured for the apply component. This user must have the necessary privileges to apply changes. This procedure grants the apply user dequeue privilege on the queue used by the apply component and configures the user as a secure queue user of the queue. In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user: • The necessary privileges to perform DML and DDL changes on the apply objects • `EXECUTE` privilege on the rule sets used by the apply component • `EXECUTE` privilege on all rule-based transformation functions used in the rule set • `EXECUTE` privilege on all apply handler procedures These privileges can be granted directly to the apply user, or they can be granted through roles. In addition, the apply user must be granted `EXECUTE` privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply component. These privileges must be granted directly to the apply user. They cannot be granted through roles. You can use the `DBMS_XSTREAM_AUTH` package to grant and revoke administrative privileges in XStream configuration. These packages do not configure the necessary privileges to perform DML or DDL changes on the apply objects. **Note:** If the apply user for an apply component is dropped using `DROP USER . . . CASCADE`, then the apply component is also dropped automatically. See "Usage Notes" for more information about this parameter.

Table 25-5 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
`apply_database_link`	The database at which the apply component applies messages. This parameter is used by an apply component when applying changes from Oracle to non-Oracle systems, such as Sybase. Set this parameter to `NULL` to specify that the apply component applies messages at the local database. **Note:** The `apply_database_link` setting cannot be altered after the apply component is created.
`apply_tag`	A binary tag that is added to redo entries generated by the specified apply component. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply component is running captures changes made by the apply component. If so, then the captured changes include the tag specified by this parameter. By default, the tag for an apply component is the hexadecimal equivalent of `'00'` (double zero). The following is an example of a tag with a hexadecimal value of `17`: `HEXTORAW('17')` If `NULL`, then the apply component generates redo entries with `NULL` tags.
`apply_captured`	Either `TRUE` or `FALSE`. If `TRUE`, then the apply component applies only the captured LCRs in the queue. Captured LCRs are LCRs that were captured by an Oracle Replication capture process. If `FALSE`, then the apply component applies only the messages in a persistent queue. These are messages that were not captured by an Oracle Replication capture process, such as persistent LCRs or user messages. To apply both captured LCRs and messages in a persistent queue, you must create at least two apply components. **Note:** The `apply_captured` setting cannot be altered after the apply component is created.

Table 25-5 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
`precommit_handler`	A user-defined procedure that can receive internal commit directives in the queue for the apply component before they are processed by the apply component. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply component. An internal commit directive is enqueued in the following ways: • When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs. • When a synchronous capture captures row LCRs, the persistent LCRs that were enqueued by the synchronous capture are organized into a message group. The synchronous capture records the transaction identifier in each persistent LCR in a transaction. • When a user or application enqueues messages and then issues a `COMMIT` statement, the commit directive is enqueued automatically. For a row LCR captured by a capture process or synchronous capture, a commit directive contains the commit SCN of the transaction from the source database. For a message enqueued by a user or application, the commit SCN is generated by the apply component. The precommit handler procedure must conform to the following restrictions: • Any work that commits must be an autonomous transaction. • Any rollback must be to a named savepoint created in the procedure. If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue. See "Usage Notes" for more information about a precommit handler procedure.
`negative_rule_set_name`	The name of the negative rule set for the apply component. The negative rule set contains the rules that instruct the apply component to discard messages. If you want to use a negative rule set for the apply component, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a negative rule set in the `hr` schema named `neg_apply_rules`, enter `hr.neg_apply_rules`. If the schema is not specified, then the current user is the default. If you specify `NULL`, and no positive rule set is specified, then the apply component applies either all captured LCRs or all of the messages in the persistent queue, depending on the setting of the `apply_captured` parameter. An error is returned if the specified rule set does not exist. If you specify both a positive and a negative rule set for an apply component, then the negative rule set is always evaluated first.

Table 25-5 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
`source_database`	The global name of the source database for the changes that will be applied by the apply component. The source database is the database where the changes originated. If an apply component applies captured messages, then the apply component can apply messages from only one capture process at one source database. If `NULL`, then the source database name of the first LCR received by the apply component is used for the source database. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify `DBS1` and the domain is `NET`, then the procedure specifies `DBS1.NET` automatically. The rules in the apply component rule sets determine which messages are dequeued by the apply component. If the apply component dequeues an LCR with a source database that is different than the source database for the apply component, then an error is raised. You can determine the source database for an apply component by querying the `DBA_APPLY_PROGRESS` data dictionary view.

Usage Notes

The following sections describe usage notes for this procedure:

- DBA Role Requirement
- Handler Procedure Names
- Message Handler and DDL Handler Procedure
- Precommit Handler Procedure
- The CREATE_APPLY Procedure and XStream Outbound Servers
- The CREATE_APPLY Procedure and XStream Inbound Servers

DBA Role Requirement

If the user who invokes this procedure is different from the user specified in the `apply_user` parameter, then the invoking user must be granted the `DBA` role. If the user who invokes this procedure is the same as the user specified in the `apply_user` parameter, then the `DBA` role is not required for the invoking user. Only the `SYS` user can set the `apply_user` to `SYS`.

Handler Procedure Names

For the `message_handler`, `ddl_handler`, and `precommit_handler` parameters, specify an existing procedure in one of the following forms:

- *`[schema_name.]procedure_name`*
- *`[schema_name.]package_name.procedure_name`*

If the procedure is in a package, then the package_name must be specified. For example, to specify a procedure in the `apply_pkg` package in the `hr` schema named `process_ddls`, enter `hr.apply_pkg.process_ddls`. An error is returned if the specified procedure does not exist.

The user who invokes the `CREATE_APPLY` procedure must have `EXECUTE` privilege on a specified handler procedure. Also, if the *`schema_name`* is not specified, then the user who invokes the `CREATE_APPLY` procedure is the default.

Message Handler and DDL Handler Procedure

The procedure specified in both the `message_handler` parameter and the `ddl_handler` parameter must have the following signature:

```
PROCEDURE handler_procedure (
   parameter_name  IN  ANYDATA);
```

Here, *`handler_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. For the message handler, the parameter passed to the procedure is a `ANYDATA` encapsulation of a user message. For the DDL handler procedure, the parameter passed to the procedure is a `ANYDATA` encapsulation of a DDL LCR.

See Also:

Logical Change Record TYPEs for information about DDL LCRs

Precommit Handler Procedure

The procedure specified in the `precommit_handler` parameter must have the following signature:

```
PROCEDURE handler_procedure (
   parameter_name   IN  NUMBER);
```

Here, *`handler_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is the commit SCN of a commit directive.

The CREATE_APPLY Procedure and XStream Outbound Servers

This procedure cannot create an XStream outbound server. To create an XStream outbound server, use the `DBMS_XSTREAM_ADM` package.

The CREATE_APPLY Procedure and XStream Inbound Servers

The following usage notes apply to this procedure and XStream inbound servers:

- The `CREATE_APPLY` procedure always creates an apply process. The apply process remains an apply process if it receives messages from a source other than an XStream client application, such as a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives messages from any other source. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.
- When creating an inbound server using the `CREATE_APPLY` procedure, set the `apply_captured` parameter to `TRUE`. Inbound servers only process LCRs captured by a capture process.
- Inbound servers can use apply handlers. Inbound servers process only DML and DDL LCRs. Therefore, inbound servers ignore message handlers specified in the `message_handler` parameter.

CREATE_OBJECT_DEPENDENCY Procedure

This procedure creates an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

An apply component schedules execution of transactions that involve the child object after all transactions with a lower commit system change number (commit SCN) that involve the parent object have been committed. An apply component uses the object identifier of the objects in the logical change records (LCRs) to detect dependencies. The apply component does not use column values in the LCRs to detect dependencies.

Note:

An error is raised if `NULL` is specified for either of the procedure parameters.

See Also:

DROP_OBJECT_DEPENDENCY Procedure

Syntax

```
DBMS_APPLY_ADM.CREATE_OBJECT_DEPENDENCY(
   object_name         IN  VARCHAR2,
   parent_object_name  IN  VARCHAR2);
```

Parameters

Table 25-6 CREATE_OBJECT_DEPENDENCY Procedure Parameters

Parameter	Description
`object_name`	The name of the child database object, specified as `[`*schema_name*`.]`*object_name*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`parent_object_name`	The name of the parent database object, specified as `[`*schema_name*`.]`*object_name*. For example, `hr.departments`. If the schema is not specified, then the current user is the default.

Usage Notes

The following usage notes apply to this procedure:

- The CREATE_OBJECT_DEPENDENCY Procedure and XStream Outbound Servers
- The CREATE_OBJECT_DEPENDENCY Procedure and XStream Inbound Servers

The CREATE_OBJECT_DEPENDENCY Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The CREATE_OBJECT_DEPENDENCY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

DELETE_ALL_ERRORS Procedure

This procedure deletes all the error transactions for the specified apply component.

Syntax

```
DBMS_APPLY_ADM.DELETE_ALL_ERRORS(
   apply_name   IN  VARCHAR2 DEFAULT NULL);
```

Parameter

Table 25-7   DELETE_ALL_ERRORS Procedure Parameter

Parameter	Description
`apply_name`	The name of the apply component that raised the errors while processing the transactions. Do not specify an owner. If `NULL`, then all error transactions for all apply components are deleted.

Usage Notes

The following usage notes apply to this procedure:

- The DELETE_ALL_ERRORS Procedure and XStream Outbound Servers
- The DELETE_ALL_ERRORS Procedure and XStream Inbound Servers

The DELETE_ALL_ERRORS Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure has no effect on XStream outbound servers.

The DELETE_ALL_ERRORS Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

DELETE_ERROR Procedure

This procedure deletes the specified error transaction.

Syntax

```
DBMS_APPLY_ADM.DELETE_ERROR(
   local_transaction_id  IN  VARCHAR2);
```

Parameter

Table 25-8   DELETE_ERROR Procedure Parameter

Parameter	Description
`local_transaction_id`	The identification number of the error transaction to delete. If the specified transaction does not exist in the error queue, then an error is raised.

Usage Notes

The following usage notes apply to this procedure:

The DELETE_ERROR Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure has no effect on XStream outbound servers.

The DELETE_ERROR Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

DROP_APPLY Procedure

This procedure drops an apply component.

Syntax

```
DBMS_APPLY_ADM.DROP_APPLY(
   apply_name             IN  VARCHAR2,
   drop_unused_rule_sets  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 25-9   DROP_APPLY Procedure Parameters

Parameter	Description
`apply_name`	The name of the apply component being dropped. You must specify an existing apply component name. Do not specify an owner.
`drop_unused_rule_sets`	If `TRUE`, then the procedure drops any rule sets, positive and negative, used by the specified apply component if these rule sets are not used by any other Oracle Replication or XStream component. These components include capture processes, propagations, apply processes, inbound servers, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If `FALSE`, then the procedure does not drop the rule sets used by the specified apply component, and the rule sets retain their rules.

Usage Notes

The following usage notes apply to this procedure:

- The DROP_APPLY Procedure and Rules
- The DROP_APPLY Procedure and XStream Outbound Servers
- The DROP_APPLY Procedure and XStream Inbound Servers

The DROP_APPLY Procedure and Rules

When you use this procedure to drop an apply component, information about rules created for the apply component is removed from the data dictionary views for rules. Information about such a rule is removed even if the rule is not in either the positive or negative rule set for the apply component. The following are the data dictionary views for rules:

- `ALL_STREAMS_GLOBAL_RULES`

- `DBA_STREAMS_GLOBAL_RULES`
- `ALL_STREAMS_SCHEMA_RULES`
- `DBA_STREAMS_SCHEMA_RULES`
- `ALL_STREAMS_TABLE_RULES`
- `DBA_STREAMS_TABLE_RULES`

The DROP_APPLY Procedure and XStream Outbound Servers

When the `DROP_APPLY` procedure is executed on an outbound server, it runs the `DROP_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package. Therefore, it might also drop the outbound server's capture process and queue.

The DROP_APPLY Procedure and XStream Inbound Servers

When the `DROP_APPLY` procedure is executed on an inbound server, it runs the `DROP_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package. Therefore, it might also drop the inbound server's queue.

DROP_OBJECT_DEPENDENCY Procedure

This procedure drops an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

Note:

- An error is raised if an object dependency does not exist for the specified database objects.
- An error is raised if `NULL` is specified for either of the procedure parameters.

See Also:

CREATE_OBJECT_DEPENDENCY Procedure

Syntax

```
DBMS_APPLY_ADM.DROP_OBJECT_DEPENDENCY(
   object_name         IN   VARCHAR2,
   parent_object_name  IN   VARCHAR2);
```

Parameters

Table 25-10 DROP_OBJECT_DEPENDENCY Procedure Parameters

Parameter	Description
`object_name`	The name of the child database object, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.

Table 25-10 (Cont.) DROP_OBJECT_DEPENDENCY Procedure Parameters

Parameter	Description
`parent_object_name`	The name of the parent database object, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.departments`. If the schema is not specified, then the current user is the default.

Usage Notes

The following usage notes apply to this procedure:

The DROP_OBJECT_DEPENDENCY Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The DROP_OBJECT_DEPENDENCY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

EXECUTE_ALL_ERRORS Procedure

This procedure re-executes the error transactions in the error queue for the specified apply component.

The transactions are re-executed in commit SCN order. Error re-execution stops if an error is raised.

Syntax

```
DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
   apply_name        IN  VARCHAR2  DEFAULT NULL,
   execute_as_user   IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 25-11 EXECUTE_ALL_ERRORS Procedure Parameters

Parameter	Description
`apply_name`	The name of the apply component that raised the errors while processing the transactions. Do not specify an owner. If `NULL`, then all error transactions for all apply components are re-executed.
`execute_as_user`	If `TRUE`, then the procedure re-executes the transactions in the security context of the current user. If `FALSE`, then the procedure re-executes each 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. The `DBA_APPLY_ERROR` data dictionary view lists the original receiver for each error transaction. The user who executes the transactions must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply component.

Usage Notes

The following usage notes apply to this procedure:

- The EXECUTE_ALL_ERRORS Procedure and XStream Outbound Servers
- The EXECUTE_ALL_ERRORS Procedure and XStream Inbound Servers

The EXECUTE_ALL_ERRORS Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

The EXECUTE_ALL_ERRORS Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

EXECUTE_ERROR Procedure

This procedure re-executes the specified error transaction in the error queue.

Syntax

```
DBMS_APPLY_ADM.EXECUTE_ERROR(
   local_transaction_id  IN  VARCHAR2,
   execute_as_user       IN  BOOLEAN    DEFAULT FALSE,
   user_procedure        IN  VARCHAR2   DEFAULT NULL);
```

Parameters

Table 25-12 EXECUTE_ERROR Procedure Parameters

Parameter	Description
`local_transaction_id`	The identification number of the error transaction to execute. If the specified transaction does not exist in the error queue, then an error is raised.
`execute_as_user`	If `TRUE`, then the procedure re-executes the transaction in the security context of the current user. If `FALSE`, then the procedure re-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. The `DBA_APPLY_ERROR` data dictionary view lists the original receiver for each error transaction. 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. This user must also have dequeue privileges on the queue used by the apply component.
`user_procedure`	A user-defined procedure that modifies the error transaction so that it can be successfully executed. Specify `NULL` to execute the error transaction without running a user procedure. **See Also:** "Usage Notes" for more information about the user procedure

Usage Notes

The following usage notes apply to this procedure:

- The User Procedure
- The EXECUTE_ERROR Procedure and XStream Outbound Servers
- The EXECUTE_ERROR Procedure and XStream Inbound Servers

The User Procedure

You must specify the full procedure name for the `user_procedure` parameter in one of the following forms:

- `[`*`schema_name`*`.]`*`package_name`*`.`*`procedure_name`*
- `[`*`schema_name`*`.]`*`procedure_name`*

If the procedure is in a package, then the *`package_name`* must be specified. The user who invokes the `EXECUTE_ERROR` procedure must have `EXECUTE` privilege on the specified procedure. Also, if the *`schema_name`* is not specified, then the user who invokes the `EXECUTE_ERROR` procedure is the default.

For example, suppose the `procedure_name` has the following properties:

- `strmadmin` is the *`schema_name`*.
- `fix_errors` is the *`package_name`*.
- `fix_hr_errors` is the *`procedure_name`*.

In this case, specify the following:

```
strmadmin.fix_errors.fix_hr_errors
```

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (
     in_anydata                    IN      ANYDATA,
     error_record                  IN      DBA_APPLY_ERROR%ROWTYPE,
     error_message_number          IN      NUMBER,
     messaging_default_processing  IN OUT  BOOLEAN,
     out_anydata                   OUT     ANYDATA);
```

The user procedure has the following parameters:

- `in_anydata`: The `ANYDATA` encapsulation of a message that the apply component passes to the procedure. A single transaction can include multiple messages. A message can be a row logical change record (row LCR), a DDL logical change record (DDL LCR), or a user message.
- `error_record`: The row in the `DBA_APPLY_ERROR` data dictionary view that identifies the transaction
- `error_message_number`: The message number of the `ANYDATA` object in the `in_anydata` parameter, starting at 1
- `messaging_default_processing`: If `TRUE`, then the apply component continues processing the message in the `in_anydata` parameter, which can include executing DML or DDL statements and invoking apply handlers.

  If `FALSE`, then the apply component skips processing the message in the `in_anydata` parameter and moves on to the next message in the `in_anydata` parameter.
- `out_anydata`: The `ANYDATA` object processed by the user procedure and used by the apply component if `messaging_default_processing` is `TRUE`.

If an LCR is executed using the `EXECUTE` LCR member procedure in the user procedure, then the LCR is executed directly, and the `messaging_default_processing` parameter should be set to `FALSE`. In this case, the LCR is not passed to any apply handlers.

Processing an error transaction with a user procedure results in one of the following outcomes:

- The user procedure modifies the transaction so that it can be executed successfully.
- The user procedure fails to make the necessary modifications, and an error is raised when transaction execution is attempted. In this case, the transaction is rolled back and remains in the error queue.

The following restrictions apply to the user procedure:

- Do not execute `COMMIT` or `ROLLBACK` statements. Doing so can endanger the consistency of the transaction.
- Do not modify `LONG`, `LONG RAW` or LOB column data in an LCR.
- If the `ANYDATA` object in the `in_anydata` parameter is a row LCR, then the `out_anydata` parameter must be row LCR if the `messaging_default_processing` parameter is set to `TRUE`.
- If the `ANYDATA` object in the `in_anydata` parameter is a DDL LCR, then the `out_anydata` parameter must be DDL LCR if the `messaging_default_processing` parameter is set to `TRUE`.
- The user who runs the user procedure must have the `SELECT` or `READ` privilege on the `DBA_APPLY_ERROR` data dictionary view.

Note:

LCRs containing transactional directives, such as `COMMIT` and `ROLLBACK`, are not passed to the user procedure.

The EXECUTE_ERROR Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

The EXECUTE_ERROR Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

GET_ERROR_MESSAGE Function

This function returns the message payload from the error queue for the specified message number and transaction identifier. The message can be a logical change record (LCR) or a non-LCR message.

This function is overloaded. One version of this function contains two `OUT` parameters. These `OUT` parameters contain the destination queue into which the message should be enqueued, if one exists, and whether the message should be executed. The destination queue is specified using the `SET_ENQUEUE_DESTINATION` procedure, and the execution directive is specified using the `SET_EXECUTE` procedure.

See Also:

- SET_ENQUEUE_DESTINATION Procedure
- SET_EXECUTE Procedure

Syntax

```
DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
   message_number           IN    NUMBER,
   local_transaction_id     IN    VARCHAR2,
   destination_queue_name   OUT   VARCHAR2,
   execute                  OUT   BOOLEAN)
RETURN ANYDATA;

DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
   message_number           IN    NUMBER,
   local_transaction_id     IN    VARCHAR2)
RETURN ANYDATA;
```

Parameters

Table 25-13 GET_ERROR_MESSAGE Function Parameters

Parameter	Description
`message_number`	The identification number of the message. This number identifies the position of the message in the transaction. Query the `DBA_APPLY_ERROR` data dictionary view to view the message number of each apply error.
`local_transaction_id`	Identifier of the error transaction for which to return a message
`destination_queue_name`	Contains the name of the queue into which the message should be enqueued. If the message should not be enqueued into a queue, then this parameter contains `NULL`.
`execute`	Contains `TRUE` if the message should be executed Contains `FALSE` if the message should not be executed

Usage Notes

The following usage notes apply to this procedure:

- The GET_ERROR_MESSAGE Procedure and XStream Outbound Servers
- The GET_ERROR_MESSAGE Procedure and XStream Inbound Servers

The GET_ERROR_MESSAGE Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

The GET_ERROR_MESSAGE Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

HANDLE_COLLISIONS

This procedure enables or disables basic conflict resolution for an apply process and a table.

Syntax

```
DBMS_APPLY_ADM.HANDLE_COLLISIONS(
   apply_name    IN  VARCHAR2,
   enable        IN  BOOLEAN,
   object        IN  VARCHAR2,
   source_object IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25-14 HANDLE_COLLISIONS Procedure Parameters

Parameter	Description
`apply_name`	The name of the apply process.
`enable`	If `TRUE`, then the following conflict resolution methods are used: • 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. • When a conflict is detected for a row that does not exist in the table, the data in the row LCR is ignored. If `FALSE` then it disables conflict resolution set by this procedure for the specified apply process and object. If `NULL`, then removes any explicit table-level setting for collision handling for the specified apply process and object.
`object`	The schema and name of the target table, specified as `[`*`schema_name`*`.]`*`table_name`* for the change of the setting. For example, if you are changing the setting for table `employees` owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.
`source_object`	The schema and object name of the source table, specified as `[`*`schema_name`*`.]`*`table_name`* for the table where the change originated. For example, if the change originated at the `employees` table owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.

SET_DML_CONFLICT_HANDLER Procedure

This procedure adds, modifies, or removes a prebuilt DML conflict handler for `INSERT`, `UPDATE`, or `DELETE` conflicts on the specified object.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Syntax

```
DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
  apply_name             IN  VARCHAR2,
  conflict_handler_name  IN  VARCHAR2,
  object                 IN  VARCHAR2  DEFAULT NULL,
  operation_name         IN  VARCHAR2  DEFAULT NULL,
  conflict_type           IN  VARCHAR2  DEFAULT NULL,
  method_name             IN  VARCHAR2  DEFAULT NULL,
  column_list             IN  VARCHAR2  DEFAULT NULL,
  resolution_column       IN  VARCHAR2  DEFAULT NULL,
  source_object           IN  VARCHAR2  DEFAULT NULL);

DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
  apply_name              IN  VARCHAR2,
  conflict_handler_name   IN  VARCHAR2,
  object                  IN  VARCHAR2  DEFAULT NULL,
  operation_name          IN  VARCHAR2  DEFAULT NULL,
  conflict_type           IN  VARCHAR2  DEFAULT NULL,
  method_name             IN  VARCHAR2  DEFAULT NULL,
  column_table            IN  DBMS_UTILITY.LNAME_ARRAY,
  resolution_column       IN  VARCHAR2  DEFAULT NULL,
  source_object           IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-15 SET_DML_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
`apply_name`	The name of the apply process.
`conflict_handler_name`	The name of the conflict handler.
`object`	The schema and name of the target table, specified as `[`*`schema name`*`.]`*`table name`* for which a conflict handler is being added, modified, or removed. For example, if an update conflict handler is being added for table `employees` owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.
`operation_name`	The name of the operation, which can be specified as: • `INSERT` • `UPDATE` • `DELETE` In order to set up conflict handlers for different operations on the same table, you must make one call per operation.
`conflict_type`	Type of update conflict handler to create. You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained: • `ROW_EXISTS`: A row with the same primary key already exists in the database for an insert or update. • `ROW_MISSING`:A row with the same primary key cannot be found for an update or delete.

Table 25-15 (Cont.) SET_DML_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
`method_name`	Type of update conflict handler to create. You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained: • `DELTA`: If the conflict type is `ROW_EXISTS` and the LCR is an update, then take the difference between the old and new values of the column in the LCR and add it to the current value of the column in the target database. The columns in the column group must be of type `NUMBER`. • `IGNORE`: Silently ignores the LCR and can be used for all conflict types. • `MAXIMUM`: Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database. This resolution method is only supported for `ROW_EXISTS` and only applies to inserts and updates. • `MINIMUM`: Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database. This resolution method is only supported for `ROW_EXISTS` and only applies to inserts and updates. • `OVERWRITE`: Applies the column list from the source database, overwriting the column values at the destination database. An `INSERT` with `ROW_EXISTS` is converted to an `UPDATE`. An `UPDATE` with `ROW_MISSING` is converted to an `INSERT`. A `DELETE` with `ROW_MISSING` is ignored. • `RECORD`: Enqueue the LCR into the error queue. Can be used for all conflict types and can only be specified for a column group that contains all the columns in the table. • `MAX_AND_EQUALS`: Applies the column list from the source database if the value of resolution column is greater than or equal to the value of the column in the database. • `MIN_AND_EQUALS`: Applies the column list from the source database if the value of resolution column is less than or equal to the value of the column in the database. If `NULL`, then the procedure removes any existing conflict handler with the same `object_name`, `resolution_group`, and `conflict_type`. If a conflict handler already exists with the same `object_name` and `resolution_column` and `conflict_type`, then the existing handler is replaced.

Table 25-15 (Cont.) SET_DML_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
`column_list`	A comma-separated list of the column names for which the conflict handler is called. The same column cannot be in more than one column list (for a given `apply_name`, `object_name`, `operation_name` and `conflict_type`). Specify `*` for the default column group, which includes all the columns in the table that are not already specified in another column list (for a given `apply_name`, `object_name`, `operation_name` and `conflict_type`). If a conflict occurs for one or more of the columns in the list when an apply component tries to apply a row logical change record (row LCR), then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list. You cannot use a `column_list` if you use a `*` in the `object_name`. The only time you can use multiple column groups is when you are specifying a conflict handler for insert or update for `ROW_EXISTS`. **Note:** Prebuilt conflict handlers do not support LOB, `LONG`, `LONG RAW`, user-defined type, and Oracle-suppled type columns. Therefore, you should not include these types of columns in the `column_list` parameter. This parameter must be set to `'*'` in the following cases: • The `operation_name` is `DELETE`. • The `method_name` is `RECORD`. • The `operation_name` is `UPDATE` and the `conflict_type` is `ROW_MISSING`.
`column_table`	An array of column names for which the conflict handler is called. This parameter is the same as the `column_list` parameter, but it uses an array instead of a list for the column names. **Note:** The `column_list` and `column_table` parameters are mutually exclusive.
`resolution_column`	For the `MAXIMUM` and `MINIMUM` prebuilt methods, the resolution column is the one tested to determine whether the current row or the LCR has the smaller value. The resolution column must be one of the columns listed in the `column_list` or `column_table` parameter. You can specify `NULL` for other resolution methods.
`source_object`	The schema and object name of the source table, specified as `[`*`schema_name`*`.]`*`table_name`* for the table where the change originated. For example, if the change originated at the `employees` table owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.

Usage Notes

The following usage notes apply to this procedure:

- Modifying an Existing Conflict Handler
- Removing an Existing Conflict Handler
- Series of Actions for Conflicts

- Procedure DML Handlers for Conflicts
- A Column Can Be in Only One Column List
- The SET_DML_CONFLICT_HANDLER Procedure and XStream Outbound Servers
- The SET_DML_CONFLICT_HANDLER Procedure and XStream Inbound Servers
- Table 25-16
- Example

Modifying an Existing Conflict Handler

If you want to modify an existing conflict handler, then you specify the `object`, `conflict_type`, and `resolution_column` of an the existing conflict handler. You can modify the `method_name` or the `column_list`.

Removing an Existing Conflict Handler

If you want to remove an existing conflict handler, then specify `NULL` for the `method_name` and specify the `object`, `conflict_type`, and `resolution_column` of the existing conflict handler.

Series of Actions for Conflicts

If an conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate conflict handler to resolve the conflict
2. If no conflict handler is specified or if the conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply component, object name, and operation name to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

> **See Also:**
>
> "Signature of a DML Handler Procedure or Error Handler Procedure" for information about setting an error handler

Procedure DML Handlers for Conflicts

If you cannot use a prebuilt conflict handler to meet your requirements, then you can create a PL/SQL procedure to use as a custom conflict handler. You use the `SET_DML_HANDLER` procedure to designate one or more custom conflict handlers for a particular table. In addition, a custom conflict handler can process LOB columns and use LOB assembly.

> **See Also:**
>
> SET_DML_HANDLER Procedure

A Column Can Be in Only One Column List

When a column is in a column list, and you try to add the same column to another column list, this procedure returns the following error:

```
ORA-00001: UNIQUE CONSTRAINT (SYS.APPLY$_CONF_HDLR_COLUMNS_UNQ1) VIOLATED
```

The `SET_DML_CONFLICT_HANDLER` Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The `SET_DML_CONFLICT_HANDLER` Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

Table 25-16 Valid Combinations of Parameters

Operation	Conflict Type	Method

`INSERT`	`ROW_EXISTS`	`OVERWRITE`
		`RECORD`
		`IGNORE`
		`MAXIMUM`
		`MINIMUM`

`UPDATE`	`ROW_EXISTS`	`OVERWRITE`
		`RECORD`
		`IGNORE`
		`MAXIMUM`
		`MINIMUM`
		`DELTA`

`UPDATE`	`ROW_MISSING`	`OVERWRITE`
		`RECORD`
		`IGNORE`

`DELETE`	`ROW_EXISTS`	`OVERWRITE`
		`RECORD`
		`IGNORE`

`DELETE`	`ROW_MISSING`	`RECORD` `IGNORE`

Example

The following is an example for setting a conflict handler for the `employees` table in the `hr` schema:

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'salary';
  cols(2) := 'commission_pct';
  DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
    apply_name            =>  'appl1',
    conflict_handler_name =>  'emp_handler_update',
    object                =>  'hr.employees',
    operation_name        =>  'UPDATE',
    conflict_type         =>  'ROW_EXISTS',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'salary',
    column_table          =>  cols);
```

```
END;
/
```

This example sets a conflict handler named `emp_handler_update` that is called if a conflict occurs for the `salary` or `commission_pct` column in the `hr.employees` table. If such a conflict occurs, then the `salary` column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the `job_id` column, then this conflict handler is not called.

SET_DML_HANDLER Procedure

This procedure sets or unsets a user procedure as a procedure DML handler for a specified operation on a specified database object for a single apply component or for all apply components in the database. The user procedure alters the apply behavior for the specified operation on the specified object.

Syntax

```
DBMS_APPLY_ADM.SET_DML_HANDLER(
   object_name          IN  VARCHAR2,
   object_type          IN  VARCHAR2,
   operation_name       IN  VARCHAR2,
   error_handler        IN  BOOLEAN   DEFAULT FALSE,
   user_procedure       IN  VARCHAR2,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL,
   apply_name           IN  VARCHAR2  DEFAULT NULL,
   assemble_lobs        IN  BOOLEAN   DEFAULT TRUE);
```

Parameters

Table 25-17 SET_DML_HANDLER Procedure Parameters

Parameter	Description
`object_name`	The name of the source object specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure.
`object_type`	The type of the source object. Currently, `TABLE` is the only possible source object type.
`operation_name`	The name of the operation, which can be specified as: • `INSERT` • `UPDATE` • `DELETE` • `LOB_UPDATE` • `DEFAULT` The procedure must be run for each operation individually. Specify `DEFAULT` to set the procedure as the default procedure DML handler for the database object. In this case, the procedure DML handler is used for any `INSERT`, `UPDATE`, `DELETE`, and `LOB_WRITE` on the database object, if another procedure DML handler is not specifically set for the operation on the database object.

Table 25-17 (Cont.) SET_DML_HANDLER Procedure Parameters

Parameter	Description
`error_handler`	If `TRUE`, then the specified user procedure is run when a row logical change record (row LCR) involving the specified operation on the specified object raises an apply error. You can code the user procedure to resolve possible error conditions, notify administrators of the error, log the error, or any combination of these actions. If `FALSE`, then the handler being set is run for all row LCRs involving the specified operation on the specified object.
`user_procedure`	A user-defined procedure that is invoked during apply for the specified operation on the specified object. If the procedure is a procedure DML handler, then it is invoked instead of the default apply performed by Oracle. If the procedure is an error handler, then it is invoked when an apply error is encountered. Specify `NULL` to unset a procedure DML handler that is set for the specified operation on the specified object.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.
`apply_name`	The name of the apply component that uses the procedure DML handler or error handler. If `NULL`, then the procedure sets the procedure DML handler or error handler as a general handler for all apply components in the database. If the `user_procedure` parameter is set to `NULL` to unset a handler, and the handler being unset is set for a specific apply component, then use the `apply_name` parameter to specify the apply component to unset the handler.
`assemble_lobs`	If `TRUE`, then LOB assembly is used for LOB columns in LCRs processed by the handler. LOB assembly combines multiple LCRs for a LOB column resulting from a single row change into one row LCR before passing the LCR to the handler. Database compatibility must be `10.2.0` or higher to use LOB assembly. If `FALSE`, then LOB assembly is not used for LOB columns in LCRs processed by the handler.

Usage Notes

The following usage notes apply to this procedure:

- Run the SET_DML_HANDLER Procedure at the Destination Database
- Procedure DML Handlers and Error Handlers
- The apply_name Parameter
- Signature of a DML Handler Procedure or Error Handler Procedure
- LOB Assembly
- The SET_DML_HANDLER Procedure and XStream Outbound Servers
- The SET_DML_HANDLER Procedure and XStream Inbound Servers

Run the SET_DML_HANDLER Procedure at the Destination Database

Run this procedure at the destination database. The `SET_DML_HANDLER` procedure provides a way for users to apply logical change records containing DML changes (row LCRs) using a customized apply.

Procedure DML Handlers and Error Handlers

If the `error_handler` parameter is set to `TRUE`, then it specifies that the user procedure is an error handler. An error handler is invoked only when a row LCR raises an apply error. Such an error can result from a data conflict if no conflict handler is specified or if the update conflict handler cannot resolve the conflict. If the `error_handler` parameter is set to `FALSE`, then the user procedure is a procedure DML handler, not an error handler, and a procedure DML handler is always run instead of performing the specified operation on the specified object.

This procedure either sets a procedure DML handler or an error handler for a particular operation on an object. It cannot set both a procedure DML handler and an error handler for the same object and operation.

Note:

Currently, setting an error handler for an apply component that is applying changes to a non-Oracle database is not supported.

The apply_name Parameter

If the `apply_name` parameter is non-`NULL`, then the procedure DML handler or error handler is set for the specified apply component. In this case, this handler is not invoked for other apply components at the local destination database. If the `apply_name` parameter is `NULL`, the default, then the handler is set as a general handler for all apply components at the destination database. When a handler is set for a specific apply component, then this handler takes precedence over any general handlers. For example, consider the following scenario:

- A procedure DML handler named `handler_hr` is specified for an apply component named `apply_hr` for `UPDATE` operations on the `hr.employees` table.
- A general procedure DML handler named `handler_gen` also exists for `UPDATE` operations on the `hr.employees` table.

In this case, the `apply_hr` apply component uses the `handler_hr` procedure DML handler for `UPDATE` operations on the `hr.employees` table.

At the source database, you must specify an unconditional supplemental log group for the columns needed by a DML or error handler.

Signature of a DML Handler Procedure or Error Handler Procedure

You can use the `SET_DML_HANDLER` procedure to set either a procedure DML handler or an error handler for row LCRs that perform a specified operation on a specified object. The signatures of a DML handler procedure and of an error handler procedure are described following this section.

In either case, you must specify the full procedure name for the `user_procedure` parameter in one of the following forms:

- `[`*`schema_name`*`.]`*`package_name.procedure_name`*
- `[`*`schema_name`*`.]`*`procedure_name`*

If the procedure is in a package, then the *`package_name`* must be specified. The user who invokes the `SET_DML_HANDLER` procedure must have `EXECUTE` privilege on the specified procedure. Also, if the *`schema_name`* is not specified, then the user who invokes the `SET_DML_HANDLER` procedure is the default.

For example, suppose the `procedure_name` has the following properties:

- `hr` is the *`schema_name`*.
- `apply_pkg` is the *`package_name`*.
- `employees_default` is the *`procedure_name`*.

In this case, specify the following:

```
hr.apply_pkg.employees_default
```

The following restrictions apply to the user procedure:

- Do not execute `COMMIT` or `ROLLBACK` statements. Doing so can endanger the consistency of the transaction that contains the LCR.
- If you are manipulating a row using the `EXECUTE` member procedure for the row LCR, then do not attempt to manipulate more than one row in a row operation. You must construct and execute manually any DML statements that manipulate more than one row.
- If the command type is `UPDATE` or `DELETE`, then row operations resubmitted using the `EXECUTE` member procedure for the LCR must include the entire key in the list of old values. The key is the primary key or the smallest unique index that has at least one `NOT NULL` column, unless a substitute key has been specified by the `SET_KEY_COLUMNS` procedure. If there is no specified key, then the key consists of all non `LOB`, non `LONG`, and non `LONG RAW` columns.
- If the command type is `INSERT`, then row operations resubmitted using the `EXECUTE` member procedure for the LCR should include the entire key in the list of new values. Otherwise, duplicate rows are possible. The key is the primary key or the smallest unique index that has at least one `NOT NULL` column, unless a substitute key has been specified by the `SET_KEY_COLUMNS` procedure. If there is no specified key, then the key consists of all of the table columns, except for columns of the following data types: 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).

The procedure specified in the `user_procedure` parameter must have the following signature:

```
PROCEDURE user_procedure (
   parameter_name   IN   ANYDATA);
```

Here, *`user_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is a `ANYDATA` encapsulation of a row LCR.

See Also:

Logical Change Record TYPEs for more information about LCRs

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (
     message            IN  ANYDATA,
     error_stack_depth  IN  NUMBER,
     error_numbers      IN  DBMS_UTILITY.NUMBER_ARRAY,
     error_messages     IN  emsg_array);
```

If you want to retry the DML operation within the error handler, then have the error handler procedure run the `EXECUTE` member procedure for the LCR. The last error raised is on top of the error stack. To specify the error message at the top of the error stack, use `error_numbers(1)` and `error_messages(1)`.

Note:

- Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.
- The `emsg_array` value must be a user-defined array that is a table of type `VARCHAR2` with at least 76 characters.

Running an error handler results in one of the following outcomes:

- The error handler successfully resolves the error and returns control to the apply component.
- The error handler fails to resolve the error, and the error is raised. The raised error causes the transaction to be rolled back and placed in the error queue.

LOB Assembly

Do not modify `LONG`, `LONG RAW`, or nonassembled LOB column data in an LCR with procedure DML handlers, error handlers, or custom rule-based transformation functions. Procedure DML handlers and error handlers can modify LOB columns in row LCRs that have been constructed by LOB assembly.

The SET_DML_HANDLER Procedure and XStream Outbound Servers

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.

The SET_DML_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

SET_ENQUEUE_DESTINATION Procedure

This procedure sets the queue where the apply component automatically enqueues a message that satisfies the specified rule.

This procedure modifies the specified rule's action context to specify the queue. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to `TRUE` for a message. In this case, the client of the rules engine is an apply component. The information in an action context is an object of type `SYS.RE$NV_LIST`, which consists of a list of name-value pairs.

A queue destination specified by this procedure always consists of the following name-value pair in an action context:

- The name is `APPLY$_ENQUEUE`.
- The value is an `ANYDATA` instance containing the queue name specified as a `VARCHAR2`.

Syntax

```
DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION(
  rule_name               IN  VARCHAR2,
  destination_queue_name  IN  VARCHAR2);
```

Parameters

Table 25-18 SET_ENQUEUE_DESTINATION Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule, specified as `[`*`schema_name`*`.]`*`rule_name`*. For example, to specify a rule named `hr5` in the `hr` schema, enter `hr.hr5` for this parameter. If the schema is not specified, then the current user is the default.
`destination_queue_name`	The name of the queue into which the apply component enqueues the message. Specify the queue in the form `[`*`schema_name`*`.]`*`queue_name`*. Only local queues can be specified. For example, to specify a queue in the `hr` schema named `streams_queue`, enter `hr.streams_queue`. If the schema is not specified, then the current user is the default. If `NULL`, then an existing name-value pair with the name `APPLY$_ENQUEUE` is removed. If no name-value pair exists with the name `APPLY$_ENQUEUE` for the rule, then no action is taken. If non-`NULL` and a name-value pair exists for the rule with the name `APPLY$_ENQUEUE`, then it is removed, and a new name-value pair with the value specified by this parameter is added.

Usage Notes

The following usage notes apply to this procedure:

- The SET_ENQUEUE_DESTINATION Procedure and Apply Handlers
- Considerations for the SET_ENQUEUE_DESTINATION Procedure
- The SET_ENQUEUE_DESTINATION Procedure and XStream Outbound Servers
- The SET_ENQUEUE_DESTINATION Procedure and XStream Inbound Servers

The SET_ENQUEUE_DESTINATION Procedure and Apply Handlers

If an apply handler, such as a procedure DML handler, DDL handler, or message handler, processes a message that also is enqueued into a destination queue, then the apply handler processes the message before it is enqueued.

Considerations for the SET_ENQUEUE_DESTINATION Procedure

The following are considerations for using this procedure:

- This procedure does not verify that the specified queue exists. If the queue does not exist, then an error is raised when an apply component tries to enqueue a message into it.
- Oracle Replication capture processes, propagations, and messaging clients ignore the action context created by this procedure.

- The apply user of the apply component using the specified rule must have the necessary privileges to enqueue messages into the specified queue. If the queue is a secure queue, then the apply user must be a secure queue user of the queue.
- The specified rule must be in the positive rule set for an apply component. If the rule is in the negative rule set for an apply component, then the apply component does not enqueue the message into the destination queue.
- If the commit SCN for a message is less than or equal to the relevant instantiation SCN for the message, then the message is not enqueued into the destination queue, even if the message satisfies the apply component rule sets.

The SET_ENQUEUE_DESTINATION Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The SET_ENQUEUE_DESTINATION Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

SET_EXECUTE Procedure

This procedure specifies whether a message that satisfies the specified rule is executed by an apply component.

This procedure modifies the specified rule's action context to specify message execution. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to `TRUE` for a message. In this case, the client of the rules engine is an apply component. The information in an action context is an object of type `SYS.RE$NV_LIST`, which consists of a list of name-value pairs.

A message execution directive specified by this procedure always consists of the following name-value pair in an action context:

- The name is `APPLY$_EXECUTE`.
- The value is an `ANYDATA` instance that contains `NO` as a `VARCHAR2`. When the value is `NO`, an apply component does not execute the message and does not send the message to any apply handler.

Syntax

```
DBMS_APPLY_ADM.SET_EXECUTE(
  rule_name   IN  VARCHAR2,
  execute     IN  BOOLEAN);
```

Parameters

Table 25-19 SET_EXECUTE Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule, specified as `[`*`schema_name`*`.]`*`rule_name`*. For example, to specify a rule named `hr5` in the `hr` schema, enter `hr.hr5` for this parameter. If the schema is not specified, then the current user is the default.

Table 25-19 (Cont.) SET_EXECUTE Procedure Parameters

Parameter	Description
`execute`	If `TRUE`, then the procedure removes the name-value pair with the name `APPLY$_EXECUTE` for the specified rule. Removing the name-value pair means that the apply component executes messages that satisfy the rule. If no name-value pair with name `APPLY$_EXECUTE` exists for the rule, then no action is taken. If `FALSE`, then the procedure adds a name-value pair to the rule's action context. The name is `APPLY$_EXECUTE` and the value is `NO`. An apply component does not execute a message that satisfies the rule and does not send the message to any apply handler. If a name-value pair exists for the rule with the name `APPLY$_EXECUTE`, then it is removed, and a new one with the value `NO` is added. If `NULL`, then the procedure raises an error.

Usage Notes

The following usage notes apply to this procedure:

- Considerations for the SET_EXECUTE Procedure
- The SET_EXECUTE Procedure and XStream Outbound Servers
- The SET_EXECUTE Procedure and XStream Inbound Servers

Considerations for the SET_EXECUTE Procedure

The following are considerations for using this procedure:

- If the message is a logical change record (LCR) and the message is not executed, then the change encapsulated in the LCR is not made to the relevant local database object. Also, if the message is not executed, then it is not sent to any apply handler.
- Oracle Replication capture processes, propagations, and messaging clients ignore the action context created by this procedure.
- The specified rule must be in the positive rule set for an apply component for the apply component to follow the execution directive. If the rule is in the negative rule set for an apply component, then the apply component ignores the execution directive for the rule.

The SET_EXECUTE Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The SET_EXECUTE Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

SET_GLOBAL_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas. This procedure overwrites any existing instantiation SCN for the database, and, if it sets the instantiation SCN for a schema or a table, then it overwrites any existing instantiation SCN for the schema or table.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. 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.

This procedure gives you precise control over which DDL logical change records (DDL LCRs) from a source database are ignored and which DDL LCRs are applied by an apply component.

Syntax

```
DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  recursive             IN  BOOLEAN   DEFAULT FALSE,
  source_root_name      IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-20 SET_GLOBAL_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
`source_database_name`	The global name of the source database. For example, `DBS1.NET`. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify `DBS1` and the domain is `NET`, then the procedure specifies `DBS1.NET` automatically.
`instantiation_scn`	The instantiation SCN. Specify `NULL` to remove the instantiation SCN metadata for the source database from the data dictionary.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.

Table 25-20 (Cont.) SET_GLOBAL_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
`recursive`	If `TRUE`, then the procedure sets the instantiation SCN for the source database, all schemas in the source database, and all tables owned by the schemas in the source database. This procedure selects the schemas and tables from the `ALL_USERS` and `ALL_TABLES` data dictionary views, respectively, at the source database under the security context of the current user. If `FALSE`, then the procedure sets the global instantiation SCN for the source database, but does not set the instantiation SCN for any schemas or tables **Note:** If `recursive` is set to `TRUE`, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the `ALL_TABLES` or `DBA_TABLES` data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.
`source_root_name`	The global name of the source root database. In a non-CDB, this parameter must be `NULL`. In a CDB, if you want to do the instantiation for a specific container than you must specify both `source_database` and `source_root_name`. If you want to do the instantiation for all the containers in the CDB, specify the `source_root_name` for the database and leave the `source_database` name as `NULL`.

Usage Notes

The following usage notes apply to this procedure:

- Instantiation SCNs and DDL LCRs
- The recursive Parameter
- Considerations for the SET_GLOBAL_INSTANTIATION_SCN Procedure
- The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Outbound Servers
- The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Inbound Servers
- The SET_GLOBAL_INSTANTIATION_SCN Procedure and CDBs

See Also:

- SET_SCHEMA_INSTANTIATION_SCN Procedure
- SET_TABLE_INSTANTIATION_SCN Procedure
- LCR$_DDL_RECORD Type for more information about DDL LCRs

Instantiation SCNs and DDL LCRs

If the commit SCN of a DDL LCR for a database object from a source database is less than or equal to the instantiation SCN for that source database at a destination database, then the apply component at the destination database disregards the DDL LCR. Otherwise, the apply component applies the DDL LCR.

The global instantiation SCN specified by this procedure is used for a DDL LCR only if the DDL LCR does not have `object_owner`, `base_table_owner`, and `base_table_name` specified. For example, the global instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of `CREATE USER`.

The recursive Parameter

If the `recursive` parameter is set to `TRUE`, then this procedure sets the instantiation SCN for each schema at a source database and for the tables owned by these schemas. This procedure uses the `SET_SCHEMA_INSTANTIATION_SCN` procedure to set the instantiation SCN for each schema, and it uses the `SET_TABLE_INSTANTIATION_SCN` procedure to set the instantiation SCN for each table. Each schema instantiation SCN is used for DDL LCRs on the schema, and each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the `recursive` parameter is set to `FALSE`, then this procedure does not set the instantiation SCN for any schemas or tables.

Considerations for the SET_GLOBAL_INSTANTIATION_SCN Procedure

The following are considerations for using this procedure:

- Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.
- The instantiation SCN is not set for the `SYS` or `SYSTEM` schemas.

The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Outbound Servers

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.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

See Also:

Oracle Database XStream Guide for more information about outbound servers and instantiation SCNs

The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Inbound Servers

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

The SET_GLOBAL_INSTANTIATION_SCN Procedure and CDBs

In a CDB, this procedure must be invoked from the same container as the apply process that uses the instantiation SCN information.

SET_KEY_COLUMNS Procedures

This procedure records the set of columns to be used as the substitute primary key for apply purposes and removes existing substitute primary key columns for the specified object if they exist.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Syntax

```
DBMS_APPLY_ADM.SET_KEY_COLUMNS(
   object_name          IN  VARCHAR2,
   column_list          IN  VARCHAR2,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL,
   apply_name           IN  VARCHAR2  DEFAULT NULL);

DBMS_APPLY_ADM.SET_KEY_COLUMNS(
   object_name          IN  VARCHAR2,
   column_table         IN  DBMS_UTILITY.NAME_ARRAY,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL,
   apply_name           IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-21 SET_KEY_COLUMNS Procedure Parameters

Parameter	Description
`object_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. If the apply component is applying changes to a non-Oracle database in a heterogeneous environment, then the object name is not verified.
`column_list`	A comma-delimited list of the columns in the table to use as the substitute primary key, with no spaces between the column names. If the `column_list` parameter is empty or `NULL`, then the current set of key columns is removed.
`column_table`	A PL/SQL associative array of type `DBMS_UTILITY.NAME_ARRAY` of the columns in the table to use as the substitute primary key. The index for `column_table` must be 1-based, increasing, dense, and terminated by a `NULL`. If the `column_table` parameter is empty or `NULL`, then the current set of key columns is removed.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.
`apply_name`	The name of the apply component.

Usage Notes

The following usage notes apply to this procedure:

- Considerations for the SET_KEY_COLUMNS Procedure
- Duplicate Rows and Substitute Primary Key Columns

- The SET_KEY_COLUMNS Procedure and XStream Outbound Servers
- The SET_KEY_COLUMNS Procedure and XStream Inbound Servers
- The SET_KEY_COLUMNS Procedure and CDBs

Considerations for the SET_KEY_COLUMNS Procedure

The following are considerations for using this procedure:

- When not empty, the specified set of columns takes precedence over any primary key for the specified object. Do not specify substitute key columns if the object has primary key columns and you want to use those primary key columns as the key.
- Run this procedure at the destination database. At the source database, you must specify an unconditional supplemental log group for the substitute key columns.
- Unlike true primary keys, columns specified as substitute key column columns can contain `NULL`s. However, Oracle recommends that each column you specify as a substitute key column be a `NOT NULL` column. You also should create a single index that includes all of the columns in a substitute key. Following these guidelines improves performance for updates, deletes, and piecewise updates to LOBs because Oracle can locate the relevant row more efficiently.
- Do not permit applications to update the primary key or substitute key columns of a table. This ensures that Oracle can identify rows and preserve the integrity of the data.
- If there is neither a primary key, nor a unique index that has at least one `NOT NULL` column, nor a substitute key for a table, then the key consists of all of the table columns, except for columns of the following data types: 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).

Duplicate Rows and Substitute Primary Key Columns

A table has duplicate rows when all of the column values are identical for two or more rows in the table, excluding LOB, `LONG`, and `LONG RAW` columns. You can specify substitute primary key columns for a table at a destination database using by the `SET_KEY_COLUMNS` procedure. When substitute primary key columns are specified for a table with duplicate rows at a destination database, and the `allow_duplicate_rows` apply component parameter is set to `Y`, meet the following requirements to keep the table data synchronized at the source and destination databases:

- Ensure that supplemental logging is specified at source database for the columns specified as substitute key columns at the destination database. The substitute key columns must be in an unconditional log group at the source database.
- Ensure that the substitute key columns uniquely identify each row in the table at the destination database.

The rest of this section provides more details about these requirements.

When there is no key for a table and the `allow_duplicate_rows` apply component parameter is set to `Y`, a single row LCR with an `UPDATE` or `DELETE` command type only is applied to one of the duplicate rows. In this case, if the table at the source database and the table at the destination database have corresponding duplicate rows, then a change that changes all of the duplicate rows at the source database also changes all the duplicate rows at the destination database when the row LCRs resulting from the change are applied.

For example, suppose a table at a source database has two duplicate rows. An update is performed on the duplicate rows, resulting in two row LCRs. At the destination database, one row LCR is applied to one of the duplicate rows. At this point, the rows are no longer duplicate at the destination database because one of the rows has changed. When the second row LCR is applied at the destination database, the rows are duplicate again. Similarly, if a delete is performed on these duplicate rows at the source database, then both rows are deleted at the destination database when the row LCRs resulting from the source change are applied.

When substitute primary key columns are specified for a table, row LCRs are identified with rows in the table during apply using the substitute primary key columns. If substitute primary key columns are specified for a table with duplicate rows at a destination database, and the `allow_duplicate_rows` apply component parameter is set to `Y`, then an update performed on duplicate rows at the source database can result in different changes when the row LCRs are applied at the destination database. Specifically, if the update does not change one of the columns specified as a substitute primary key column, then the same duplicate row can be updated multiple times at the destination database, while other duplicate rows might not be updated.

Also, if the substitute key columns do not identify each row in the table at the destination database uniquely, then a row LCR identified with multiple rows can update any one of the rows. In this case, the update in the row LCR might not be applied to the correct row in the table at the destination database.

An apply component ignores substitute primary key columns when it determines whether rows in a table are duplicates. An apply component determines that rows are duplicates only if all of the column values in the rows are identical (excluding LOB, `LONG`, and `LONG RAW` columns). Therefore, an apply component always raises an error if a single update or delete changes two or more nonduplicate rows in a table.

For example, consider a table with columns `c1`, `c2`, and `c3` on which the `SET_KEY_COLUMNS` procedure is used to designate column `c1` as the substitute primary key. If two rows have the same key value for the `c1` column, but different value for the `c2` or `c3` columns, then an apply component does not treat the rows as duplicates. If an update or delete modifies more than one row because the `c1` values in the rows are the same, then the apply component raises an error regardless of the setting for the `allow_duplicate_rows` apply component parameter.

See Also:

SET_PARAMETER Procedure for more information about the `allow_duplicate_rows` apply component parameter

The SET_KEY_COLUMNS Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The SET_KEY_COLUMNS Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

The SET_KEY_COLUMNS Procedure and CDBs

This procedure defines the columns that are used as a substitute primary key. You must perform the `SET_KEY_COLUMNS` procedure in the appropriate PDB.

SET_PARAMETER Procedure

This procedure sets an apply parameter to the specified value.

Syntax

```
DBMS_APPLY_ADM.SET_PARAMETER (
   apply_name   IN  VARCHAR2,
   parameter    IN  VARCHAR2,
   value        IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-22 SET_PARAMETER Procedure Parameters

Parameter	Description
`apply_name`	The apply component name. Do not specify an owner.
`parameter`	The name of the parameter you are setting.
`value`	The value to which the parameter is set. If `NULL`, then the parameter is set to its default value.

Apply Component Parameters

The following table lists the parameters for an apply component.

Note:

Starting from Oracle Database 21c release, `OPTIMIZE_PROGRESS_TABLE` is desupported.

Table 25-23 Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`allow_duplicate_rows`	`Y` or `N`	`N`	If `Y` and more than one row is changed by a single row logical change record (row LCR) with an `UPDATE` or `DELETE` command type, then the apply component only updates or deletes one of the rows. If `N`, then the apply component raises an error when it encounters a single row LCR with an `UPDATE` or `DELETE` command type that changes more than one row in a table. **Note:** Regardless of the setting for this parameter, apply components do not allow changes to duplicate rows for tables with LOB, `LONG`, or `LONG RAW` columns. **See Also:** "Usage Notes" and "Duplicate Rows and Substitute Primary Key Columns"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`apply_sequence_nextval`	`Y` or `N`	`N` for apply processes `Y` for XStream outbound servers and XStream inbound servers	Controls whether the apply component checks and adjusts sequence values. If `Y`, then the apply component checks and adjusts sequence values. For ascending sequences, setting this parameter to `Y` ensures that the destination sequence values are equal to or greater than the source sequence values. For descending sequences, setting this parameter to `Y` ensures that the destination sequence values are equal to or less than the source sequence values. If `N`, then the apply component does not check or adjust sequence values. **Note:** This parameter is intended for XStream. Do not set this parameter to `Y` for an apply process in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations. **See Also:** SET_PARAMETER Procedure for information about the `capture_sequence_nextval` capture process parameter

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`batchsql_mode`	`DEPENDENT`, `DEPENDENT_EAGER`, or `SEQUENTIAL`	`DEPENDENT`	Determines the batching method used to generate batch transactions for reordering. This parameter can be set to one of the following: • `DEPENDENT` - Batch transactions in a dependency-aware manner to minimize cross-batch dependencies and improve parallel processing performance in Oracle GoldenGate `BATCHSQL` mode. An executing batch has no unresolved dependencies. • `DEPENDENT_EAGER` - Batch transactions in a dependency-aware manner to minimize cross-batch dependencies and improve parallel processing performance in Oracle GoldenGate `BATCHSQL` mode. A batch can be executed when there are unresolved dependencies. The apply server waits for dependencies to be resolved before executing an LCR. • `SEQUENTIAL` - Batch transactions in a sequential manner. `COMMIT_SERIALIZATION = FULL` runs in this mode regardless of the Oracle GoldenGate `BATCHSQL` mode. **Note:** This parameter is intended for Oracle GoldenGate. Do not use this parameter in an Oracle Replication environment or in an XStream environment.
`cdgranularity`	`ROW` or `COLGROUP`	`COLGROUP` for XStream In `ROW` for Oracle GoldenGate	Specifies the conflict detection granularity. This parameter can be set to one of the following: • `ROW` - Conflict resolution is applied for all column groups if there is a conflict for any column group. • `COLGROUP` - Conflict resolution is applied for column groups that have a conflict. Conflict resolution is not applied for column groups that do not have a conflict. **Note:** This parameter is intended for XStream and Oracle GoldenGate. Do not set this parameter for an apply process in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations.

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`commit_serialization`	`DEPENDENT_TRANSACTIONS` or `FULL`	`DEPENDENT_TRANSACTIONS`	The order in which applied transactions are committed. Apply servers can apply nondependent transactions at the destination database in an order that is different from the commit order at the source database. Dependent transactions are always applied at the destination database in the same order as they were committed at the source database. You control whether the apply servers can apply nondependent transactions in a different order at the destination database using the `commit_serialization` apply parameter. This parameter has the following settings: • `DEPENDENT_TRANSACTIONS` - The apply component can commit nondependent transactions in any order. Performance is best if you specify `DEPENDENT_TRANSACTIONS`. • `FULL` - The apply component commits applied transactions in the order in which they were committed at the source database. Regardless of the specification, applied transactions can execute in parallel subject to data dependencies and constraint dependencies. If you specify `DEPENDENT_TRANSACTIONS`, then a destination database might commit changes in a different order than the source database. For example, suppose two nondependent transactions are committed at the source database in the following order: 1. Transaction A 2. Transaction B At the destination database, these transactions might be committed in the opposite order: 1. Transaction B 2. Transaction A If you specify `DEPENDENT_TRANSACTIONS` and there are application constraints that are not enforced by the database, then use virtual dependency definitions or add `RELY` constraints to account for the application constraints. See *Oracle Database Data Warehousing Guide* for information about `RELY` constraints. **Note:** The `NONE` value is deprecated for this parameter. It is replaced by the `DEPENDENT_TRANSACTIONS` value.

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
			See Also: "Usage Notes"
`compare_key_only`	`Y` or `N`	`N` for apply processes `Y` for XStream inbound servers	If `Y`, then disables automatic conflict detection and only uses primary and unique key columns to identify the table row for a row LCR. If `N`, then enables automatic conflict detection and uses all of the old values in a row LCR to identify the table row for a row LCR. **Note:** The `COMPARE_OLD_VALUES` procedure in this package can disable comparison of old values for specified columns during apply. See COMPARE_OLD_VALUES Procedure. **See Also:** "Usage Notes"
`compute_lcr_dep_on_arrival`	`Y` or `N`	`N`	If `Y`, the dependencies are computed as the LCRs for the transaction are received. If `N`, the dependencies are computed only after all the LCRs for a transaction are received. If the target table has all of the same constraints as the source table, you can improve the performance by setting this parameter to `Y`. If the number of LCRs in transaction exceeds the value of the number of the `eager_size` parameter, then the dependencies for that transaction are calculated on arrival regardless of the setting of `compute_lcr_dep_on_arrival`. **Note:** This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations.
`disable_on_error`	`Y` or `N`	`Y`	If `Y`, then the apply component is disabled on the first unresolved error, even if the error is not irrecoverable. If `N`, then the apply component continues regardless of unresolved errors.

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`disable_on_limit`	`Y` or `N`	`N`	If `Y`, then the apply component is disabled if the apply component terminates because it reached a value specified by the `time_limit` parameter or `transaction_limit` parameter. If `N`, then the apply component is restarted immediately after stopping because it reached a limit. When an apply component is restarted, it gets a new session identifier, and the processes associated with the apply component also get new session identifiers. However, the coordinator process number (`AP`*`nn`*) remains the same.
`eager_size`	A positive integer	`9500`	The apply component usually waits until it receives a commit record before starting to apply changes of a transaction. If XStream is enabled and more than `eager_size` LCRs arrive for a given transaction, then apply starts processing the changes. If XStream is not enabled and more than `eager_size` LCRs arrive for a given transaction, then apply waits until the complete transaction is received before processing the changes. Since it is possible that all existing apply servers are handling complete transactions from the source, additional apply servers are automatically created to handle outstanding eager transactions. The apply parameter `max_parallelism` limits the maximum number of apply servers that can be used for an apply process. This apply parameter is relevant only if its value is less than the value of the `txn_lcr_spill_threshold` apply parameter. When the value of `txn_lcr_spill_threshold` is lower than the value of `eager_size`, transactions spill to disk before eager apply begins. **Note:** This parameter is intended for XStream. Do not set this parameter in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations.

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`enable_xstream_table_stats`	`Y` or`N`	`Y`	When this parameter is set to `Y`, statistics about the operations of applied transactions are collected and made available in the `V$XSTREAM_TABLE_STATS` view. When this parameter is set to `N`, no statistics are collected. **Note:** This parameter is intended for XStream. Do not set this parameter to `Y` for an apply process in an Oracle Replication environment.
`excludetag`	Comma-delimited list of Oracle Replication tags	`NULL`	Controls whether the capture process for an outbound server captures DML changes that are tagged with one of the specified Oracle Replication tags. Whether the capture process captures these changes depends on the settings for the `getapplops` and `getreplicates` parameters. If `NULL`, then the capture process ignores this parameter. **Note:** This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes" for the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure for more information about this parameter
`excludetrans`	Comma-delimited list of transaction names	`NULL`	Controls whether the capture process for an outbound server captures DML changes in the specified transaction names. Whether the capture process captures these changes depends on the settings for the `getapplops` and `getreplicates` parameters. If `NULL`, then the capture process ignores this parameter. **Note:** This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes" for the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure for more information about this parameter

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`excludeuser`	Comma-delimited list of user names	`NULL`	Controls whether the capture process for an outbound server captures DML changes made by the specified users. Whether the capture process captures these changes depends on the settings for the `getapplops` and `getreplicates` parameters. Specify an exact pattern match for each user name. The pattern match is case sensitive. For example, specify `HR` for the `hr` user. If `NULL`, then the capture process ignores this parameter. **Note:** This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes" for the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure for more information about this parameter
`excludeuserid`	Comma-delimited list of user ID values	`NULL`	Controls whether the capture process for an outbound server captures data manipulation language (DML) changes made by the specified users. Whether the capture process captures these changes depends on the settings for the `getapplops` and `getreplicates` parameters. To view the user ID for a user, query the `USER_ID` column in the `ALL_USERS` data dictionary view. If `NULL`, then the capture process ignores this parameter. **Note:** This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes" for the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure for more information about this parameter

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`getapplops`	`Y` or `N`	`Y`	If `Y`, then the capture process captures DML changes if the original user is not specified in the `excludeuserid` or `excludeuser` parameters and the transaction name is not specified in the `excludetrans` parameter. If `N`, then the capture process ignores DML changes if the original user is not specified in the `excludeuserid` or `excludeuser` parameters and the transaction name is not specified in the `excludetrans` parameter. In either case, the capture process captures a DML change only if it satisfies the capture process's rule sets. When `N` is set for both `getapplops` and `getreplicates`, no data is captured. **Note:** This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes" for the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure for more information about this parameter
`getreplicates`	`Y` or `N`	`N`	If `Y`, then the capture process captures DML changes if the original user is specified in the `excludeuserid` or `excludeuser` parameters and the transaction name is specified in the `excludetrans` parameter. If `N`, then the capture process ignores DML changes if the original user is specified in the `excludeuserid` or `excludeuser` parameters and the transaction name is specified in the `excludetrans` parameter. In either case, the capture process captures a DML change only if it satisfies the capture process's rule sets. When `N` is set for both `getapplops` and `getreplicates`, no data is captured. **Note:** This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes" for the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure for more information about this parameter

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`grouptransops`	A positive integer from `1` to `10000`	`250` for apply processes and XStream inbound servers `10000` for XStream outbound servers	The minimum number of LCRs that can be grouped into a single transaction. The commit LCR for a transaction is not included in the LCR count for the transaction. This parameter enables an apply component to group LCRs from multiple transactions into a single transaction. The apply component groups only LCRs that are part of committed transactions. If a transaction has more LCRs than the setting for this parameter, then the transaction is applied as a single transaction. The apply component does not split a transaction into separate transactions. This parameter only takes effect if the `parallelism` parameter setting is `1`. The `grouptransops` parameter is ignored if the `parallelism` parameter setting is greater than `1`. **Note:** This parameter is intended for XStream outbound servers and inbound servers. An Oracle Apply process ignores this parameter unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`handlecollisions`	`Y` or `N`	`N`	This parameter controls whether the apply component tries to resolve duplicate-record and missing-record errors when applying changes during data loading. This parameter should be set to `N` for normal replication activity. It should be set to `Y` only when data is being loaded (instantiated) and replication is enabled. If `Y`, then does the equivalent of `OVERWRITE` for `INSERT`, `UPDATE` and `DELETE` operations that get `ROW_EXISTS` errors, and ignores `UPDATE` and `DELETE` operations that get `ROW_MISSING` errors. Specifically, the apply component performs the following actions when this parameter is set to `Y`: • If the operation is an insert and the primary key or unique key exists, then the insert is converted to an update. • If the operation is an update that does not modify the primary key or unique key columns, and the row does not exist, then the change is ignored. • If the operation is an update that modifies the primary key or unique key columns, and the row does not exist, then the change is ignored. • If the operation is an update that modifies the primary key or unique key columns, and the a row with the new key values already exists, then delete the row with the old key values and replace the row with the new key values. • If the operation is a delete and the row does not exist, then the change is ignored. If `N` then it disables the above settings. **Note:** This parameter is intended for an XStream In environment with one or more inbound servers. Do not set this parameter in an Oracle Replication environment.

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`ignore_transaction`	A valid source transaction ID or `NULL`	`NULL`	Instructs the apply component to ignore the specified transaction from the source database, effective immediately. Use caution when setting this parameter because ignoring a transaction might lead to data divergence between the source database and destination database. To ignore multiple transactions, specify each transaction in a separate call to the `SET_PARAMETER` procedure. The `DBA_APPLY_PARAMETERS` view displays a comma-delimited list of all transactions to be ignored. To clear the list of ignored transactions, run the `SET_PARAMETER` procedure and specify `NULL` for the `ignore_transaction` parameter. If `NULL`, then the apply component ignores this parameter. **Note:** An apply component ignores this parameter for transactions that were not captured by a capture process. **See Also:** "Usage Notes"
`maximum_scn`	A valid SCN or `INFINITE`	`INFINITE`	The apply component is disabled before applying a transaction with a commit SCN greater than or equal to the value specified. If `INFINITE`, then the apply component runs regardless of the SCN value. **Note:** An apply component ignores this parameter for transactions that were not captured by a capture process. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`max_parallelism`	A positive integer	`50`	Limits the maximum number of apply servers that can be used for an apply component. When the apply parallelism parameter is set greater than one, the apply component adds apply servers when necessary to process transactions until it reaches the limit set by this parameter (`max_parallelism`). Transactions include both unassigned (eager) and assigned transactions. Apply servers that idle for more than 5 minutes are shut down until the configured `parallelism` is attained. Runtime statistics for servers that have been shut down are aggregated into apply server 0 so that accurate apply statistics for the entire apply process can be maintained. **Note:** This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations.
`max_sga_size`	A positive integer	`INFINITE`	Controls the amount of system global area (SGA) memory allocated specifically to the apply component, in megabytes. The memory is allocated for the duration of the apply component's session and is released when the apply component becomes disabled. **Note:** 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. If `NULL`, then the apply component uses the original default value. A `NULL` value has the same effect as resetting the parameter to its default value. **Note:** This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`message_tracking_frequency`	`0` or a positive integer	`2000000`	The frequency at which messages applied by the inbound server are tracked automatically. For example, if this parameter is set to the default value of `2000000`, then every two-millionth message is tracked automatically. The tracking label used for automatic message tracking is *`inbound_server_name`*`:AUTOTRACK`, where *`inbound_server_name`* is the name of the inbound server. Only the first 20 bytes of the inbound server name are used; the rest is truncated if it exceeds 20 bytes. If `0` (zero), then no messages are tracked automatically.
`optimize_self_updates`	`Y` or `N`	`Y`	This parameter affects conflict resolution when an update in the source database sets a column to its existing value. When this parameter is set to `Y`, a conflict between the value in the LCR and the corresponding column in the target database is considered resolved. When this parameter is set to `N`, the conflict is processed.
`parallelism`	A positive integer	`4`	The number of apply servers that can concurrently apply transactions. The reader server and the apply server process names are `AS`*`nn`*, where *`nn`* can include letters and numbers. The total number of `AS`*`nn`* processes is the value of the `parallelism` parameter plus one. For example, if parallelism is set to `4`, then an apply component uses a total of five `AS`*`nn`* processes. In this case, there is one reader server and four apply servers. Setting the `parallelism` parameter to a number higher than the number of available operating system user processes can disable the apply component. Make sure the `PROCESSES` initialization parameter is set appropriately when you set the `parallelism` parameter. **Note:** When the value of this parameter is changed from `1` to a higher value for a running apply component, the apply component is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied. When the value of this parameter is greater than `1`, and the parameter value is decreased or increased, the apply component does not restart. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`parallelism_interval`	`0` or a positive integer	`0`	The parallelism interval is the interval in seconds at which the current workload activity is computed. The apply component calculates the mean throughput every 5 X `parallelism_interval` seconds. After each calculation, the apply component can increase or decrease the number of apply servers to try to improve throughput. If throughput is improved, then the apply component keeps the new number of apply servers. The parallelism interval is used only if the `parallelism` parameter is set to a value greater than one and the `max_parallelism` parameter value is greater than the `parallelism` parameter value. **Note:** This parameter is intended for an XStream In environment with one or more inbound servers. XStream outbound servers ignore this parameter. Do not set this parameter in an Oracle Replication environment unless XStream optimizations are enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`preserve_encryption`	`Y` or `N`	`Y`	Whether to preserve encryption for columns encrypted using Transparent Data Encryption. If `Y`, then columns in tables at the destination database must be encrypted when corresponding columns in row LCRs are encrypted. If columns are encrypted in row LCRs but the corresponding columns are not encrypted in the tables at the destination database, then an error is raised when the apply component tries to apply the row LCRs. If `N`, then columns in tables at the destination database do not need to be encrypted when corresponding columns in row LCRs are encrypted. If columns are encrypted in row LCRs but the corresponding columns are not encrypted in the tables at the destination database, then the apply component applies the changes in the row LCRs. **Note:** When the value of this parameter is changed for a running apply component, the apply component is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied. **See Also:** "Usage Notes"
`rtrim_on_implicit_conversion`	`Y` or `N`	`Y`	Whether to remove blank padding from the right end of a column when automatic data type conversion is performed during apply. If `Y`, then blank padding is removed when a `CHAR` or `NCHAR` source column in a row LCR is converted to a `VARCHAR2`, `NVARCHAR2`, or `CLOB` column in a table. If `N`, then blank padding is preserved in the column. **See Also:** "Usage Notes"
`startup_seconds`	`0`, a positive integer, or `INFINITE`	`0`	The maximum number of seconds to wait for another instantiation of the same apply component to finish. If the other instantiation of the same apply component does not finish within this time, then the apply component does not start. If `INFINITE`, then an apply component does not start until another instantiation of the same apply component finishes. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`suppresstriggers`	`Y` or `N`	`Y`	This parameter controls whether triggers fire when a change is made by the apply component. If `Y`, triggers do not fire for changes made by the apply component. If `N`, triggers fire for changes made by the apply component. If a trigger's firing property is set to always fire, then the trigger always fires for changes made by the apply component, regardless of the value of the `suppresstriggers` parameter. A trigger's firing property is set to always fire by running the `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY` procedure with the `fire_once` parameter set to `FALSE`. **Note:** This parameter is intended for an XStream In environment with one or more inbound servers. Do not set this parameter in an Oracle Replication environment. **See Also:** "Usage Notes"
`time_limit`	A positive integer or `INFINITE`	`INFINITE`	The apply component stops as soon as possible after the specified number of seconds since it started. If `INFINITE`, then the apply component continues to run until it is stopped explicitly. **See Also:** "Usage Notes"
`trace_level`	`0` or a positive integer	`0`	Set this parameter only under the guidance of Oracle Support Services. **See Also:** "Usage Notes"
`transaction_limit`	A positive integer or `INFINITE`	`INFINITE`	The apply component stops after applying the specified number of transactions. If `INFINITE`, then the apply component continues to run regardless of the number of transactions applied. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`txn_age_spill_threshold`	A positive integer or `INFINITE`	`900`	The apply component begins to spill messages from memory to hard disk for a particular transaction when the amount of time that any message in the transaction has been in memory exceeds the specified number. The parameter specifies the age in seconds. When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table. Message spilling occurs at the transaction level. For example, if this parameter is set to `900`, and the reader server of an apply component detects that one message in a transaction has been in memory longer than 900 seconds, then all of the messages in the transaction spill from memory to hard disk. If `INFINITE`, then the apply component does not spill messages to the hard disk based on the age of the messages. Query the `DBA_APPLY_SPILL_TXN` data dictionary view for information about transactions spilled by an apply component. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`txn_lcr_spill_threshold`	A positive integer or `INFINITE`	`10000`	The apply component begins to spill messages from memory to hard disk for a particular transaction when the number of messages in memory for the transaction exceeds the specified number. The number of messages in first chunk of messages spilled from memory equals the number specified for this parameter, and the number of messages spilled in future chunks is either 100 or the number specified for this parameter, whichever is less. If the reader server of an apply component has the specified number of messages in memory for a particular transaction, then when it detects the next message for this transaction, it spills the messages that are in memory to the hard disk. For example, if this parameter is set to `10000`, and a transaction has 10,200 messages, then the reader server handles the transaction in the following way: 1. Reads the first 10,000 messages in the transaction into memory 2. Spills messages 1 - 10,000 to hard disk when it detects message 10,000 3. Reads the next 100 messages in the transaction into memory 4. Spills messages 10,001 - 10,100 to hard disk when it detects message 10,100 5. Reads the next 100 messages in the transaction into memory The apply component applies the first 10,100 messages from the hard disk and the last 100 messages from memory. When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table. Message spilling occurs at the transaction level. For example, if this parameter is set to `10000`, and the reader server of an apply component is assembling two transactions, one with 7,500 messages and another with 8,000 messages, then it does not spill any messages. If `INFINITE`, then the apply component does not spill messages to the hard disk based on the number of messages in a transaction. Query the `DBA_APPLY_SPILL_TXN` data dictionary view for information about transactions spilled by an apply component. **See Also:** "Usage Notes"

Table 25-23 (Cont.) Apply Component Parameters

Parameter Name	Possible Values	Default	Description
`write_alert_log`	`Y` or `N`	`Y`	If `Y`, then the apply component writes a message to the alert log on exit. If `N`, then the apply component does not write a message to the alert log on exit. The message specifies the reason why the apply component stopped.

Usage Notes

The following usage notes apply to this procedure:

- Delays Are Possible Before New Parameter Settings Take Effect
- Parameters Interpreted as Positive Integers
- Parameters with a System Change Number (SCN) Setting
- The SET_PARAMETER Procedure and Replication
- The SET_PARAMETER Procedure and XStream Outbound Servers
- The SET_PARAMETER Procedure and XStream Inbound Servers

Delays Are Possible Before New Parameter Settings Take Effect

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

Parameters Interpreted as Positive Integers

For all parameters that are interpreted as positive integers, the maximum possible value is `4,294,967,295`. Where applicable, specify `INFINITE` for larger values.

Parameters with a System Change Number (SCN) Setting

For parameters that require an SCN setting, any valid SCN value can be specified.

The SET_PARAMETER Procedure and Replication

You can use the following parameters in Replication if you enable XStream performance optimizations for Oracle Replication using the procedure `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS`:

- `apply_sequence_nextval`
- `compute_lcr_dep_on_arrival`
- `eager_size`
- `grouptransops`
- `max_parallelism`
- `max_sga_size`
- `parallelism_interval`

The SET_PARAMETER Procedure and XStream Outbound Servers

Outbound servers ignore the settings for the following apply parameters:

- `allow_duplicate_rows`
- `commit_serialization`
- `compare_key_only`
- `compute_lcr_dep_on_arrival`
- `disable_on_error`
- `eager_size`
- `enable_xstream_table_stats`
- `grouptransops`
- `handlecollisions`
- `optimize_self_updates`
- `parallelism`
- `parallelism_interval`
- `preserve_encryption`
- `rtrim_on_implicit_conversion`
- `suppresstriggers`

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.

You can use the other apply parameters with outbound servers.

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

The SET_PARAMETER Procedure and XStream Inbound Servers

Inbound servers ignore the settings for the following apply parameters:

- `excludetag`
- `excludetrans`
- `excludeuser`
- `excludeuserid`
- `getapplops`
- `getreplicates`
- `ignore_transaction`
- `maximum_scn`

You can use all of the other apply component parameters with inbound servers.

The default setting for the `compare_key_only` parameter for an inbound server is `Y`.

The default setting for the `parallelism` parameter for an inbound server is `4`.

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

SET_REPERROR_HANDLER Procedure

This procedure specifies how a particular error is handled based on its error number.

You can choose between several predefined actions for a given error.

Syntax

```
DBMS_APPLY_ADM.SET_REPERROR_HANDLER(
  apply_name     IN  VARCHAR2,
  object         IN  VARCHAR2,
  error_number   IN  NUMBER,
  method         IN  VARCHAR2,
  source_object  IN  VARCHAR2   DEFAULT NULL,
  max_retries    IN  NUMBER     DEFAULT NULL,
  delay_csecs    IN  NUMBER     DEFAULT 6000);
```

Parameters

Table 25-24 SET_REPERROR_HANDLER Procedure Parameters

Parameter	Description
`apply_name`	The name of the apply process.
`object`	The schema and name of the target table, specified as `[`*`schema name`*`.]`*`table name`* for which an error handler is being added, modified, or removed. The table must exist. For example, if an update conflict handler is being added for table `employees` owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.
`error_number`	The error handling number. If `0`, then use the default for all error handling for `object`.
`method`	Specifies the action to take when the given `error_number` occurs. If `NULL`, remove the error handler for `error_number` The possible actions are: • `ABEND`: Stop the apply process when the error occurs. • `RECORD`: Move the LCR to the error queue when the error is encountered. • `IGNORE`: Silently ignore the error and do not apply the LCR. • `RETRY`: Retry the LCR `max_retries` times. • `RETRY_TRANSACTION`: Retry the transaction `max_retries` times. Wait `delay_csecs` centiseconds before each retry. `RECORD_TRANSACTION` : Move the entire transaction to the error queue if this error occurs.

Table 25-24 (Cont.) SET_REPERROR_HANDLER Procedure Parameters

Parameter	Description
`source_object`	The schema and object name of the source table, specified as `[schema_name.]table_name` for the table where the change originated. For example, if the change originated at the `employees` table owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.
`max_retires`	Maximum number of times to retry for `RETRY` and `RETRY_TRANSACTION` actions in `method`. Must be specified with either the `RETRY` or `RETRY_TRANSACTION`
`delay_csecs`	The number of centiseconds between retries for `RETRY` and `RETRY_TRANSACTION` action in `method`.

Usage Notes

The following usage notes apply to this procedure:

- Priority of Error Handlers

Priority of Error Handlers

Any conflict handling specified by `SET_UPDATE_CONFLICT_HANDLER` or `SET_DML_CONFLICT_HANDLER` is tried before the actions specified by `SET_REPERROR_HANDLER`. The PL/SQL procedure specified by `SET_DML_HANDLER` is called to handle the error if none of the previously mentioned methods resolve it.

SET_SCHEMA_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database. This procedure overwrites any existing instantiation SCN for the schema, and, if it sets the instantiation SCN for a table, it overwrites any existing instantiation SCN for the table.

This procedure gives you precise control over which DDL logical change records (LCRs) for a schema are ignored and which DDL LCRs are applied by an apply component.

Syntax

```
DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN(
  source_schema_name    IN  VARCHAR2,
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  recursive             IN  BOOLEAN   DEFAULT FALSE,
  source_root_name      IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-25 SET_SCHEMA_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
`source_schema_name`	The name of the source schema. For example, `hr`. When setting an instantiation SCN for schema, always specify the name of the schema at the source database, even if a rule-based transformation or apply handler is configured to change the schema name.
`source_database_name`	The global name of the source database. For example, `DBS1.NET`. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify `DBS1` and the domain is `NET`, then the procedure specifies `DBS1.NET` automatically.
`instantiation_scn`	The instantiation SCN. Specify `NULL` to remove the instantiation SCN metadata for the source schema from the data dictionary.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.
`recursive`	If `TRUE`, then the procedure sets the instantiation SCN for the specified schema and all tables owned by the schema in the source database. This procedure selects the tables owned by the specified schema from the `ALL_TABLES` data dictionary view at the source database under the security context of the current user. If `FALSE`, then the procedure sets the instantiation SCN for specified schema, but does not set the instantiation SCN for any tables **Note:** If `recursive` is set to `TRUE`, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the `ALL_TABLES` or `DBA_TABLES` data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.
`source_root_name`	The global name of the source root database. In a non-CDB, this parameter must be `NULL`. In a CDB, both `source_database` and `source_root_name` must be specified to identify a specific container.

Usage Notes

The following usage notes apply to this procedure:

- The SET_SCHEMA_INSTANTIATION_SCN Procedure and LCRs
- Instantiation SCNs and DDL LCRs
- The recursive Parameter
- The SET_SCHEMA_INSTANTIATION_SCN Procedure and XStream Outbound Servers
- The SET_SCHEMA_INSTANTIATION_SCN Procedure and XStream Inbound Servers
- The SET_SCHEMA_INSTANTIATION_SCN Procedure and CDBs

See Also:

- SET_GLOBAL_INSTANTIATION_SCN Procedure
- SET_TABLE_INSTANTIATION_SCN Procedure
- LCR$_DDL_RECORD Type for more information about DDL LCRs

The SET_SCHEMA_INSTANTIATION_SCN Procedure and LCRs

Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

Instantiation SCNs and DDL LCRs

If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at a destination database, then the apply component at the destination database disregards the DDL LCR. Otherwise, the apply component applies the DDL LCR.

The schema instantiation SCN specified by this procedure is used on the following types of DDL LCRs:

- DDL LCRs with a `command_type` of `CREATE TABLE`
- DDL LCRs with a non-`NULL` `object_owner` specified and neither `base_table_owner` nor `base_table_name` specified.

For example, the schema instantiation SCN set by this procedure is used for a DDL LCR with a `command_type` of `CREATE TABLE` and `ALTER USER`.

The schema instantiation SCN specified by this procedure is not used for DDL LCRs with a `command_type` of `CREATE USER`. A global instantiation SCN is needed for such DDL LCRs.

The recursive Parameter

If the `recursive` parameter is set to `TRUE`, then this procedure sets the table instantiation SCN for each table at the source database owned by the schema. This procedure uses the `SET_TABLE_INSTANTIATION_SCN` procedure to set the instantiation SCN for each table. Each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the `recursive` parameter is set to `FALSE`, then this procedure does not set the instantiation SCN for any tables.

The SET_SCHEMA_INSTANTIATION_SCN Procedure and XStream Outbound Servers

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.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

See Also:

Oracle Database XStream Guide for more information about outbound servers and instantiation SCNs

The SET_SCHEMA_INSTANTIATION_SCN Procedure and XStream Inbound Servers

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

The SET_SCHEMA_INSTANTIATION_SCN Procedure and CDBs

In a CDB, this procedure must be invoked from the same container as the apply process that uses the instantiation SCN information.

SET_TABLE_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified table in the specified source database. This procedure overwrites any existing instantiation SCN for the particular table.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. 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.

This procedure gives you precise control over which logical change records (LCRs) for a table are ignored and which LCRs are applied by an apply component.

Syntax

```
DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
  source_object_name     IN  VARCHAR2,
  source_database_name   IN  VARCHAR2,
  instantiation_scn      IN  NUMBER,
  apply_database_link    IN  VARCHAR2  DEFAULT NULL,
  source_root_name       IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 25-26 SET_TABLE_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
`source_object_name`	The name of the source object specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. When setting an instantiation SCN for a database object, always specify the name of the schema and database object at the source database, even if a rule-based transformation or apply handler is configured to change the schema name or database object name.
`source_database_name`	The global name of the source database. For example, `DBS1.NET`. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify `DBS1` and the domain is `NET`, then the procedure specifies `DBS1.NET` automatically.
`instantiation_scn`	The instantiation SCN. Specify `NULL` to remove the instantiation SCN metadata for the source table from the data dictionary.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database. **Note:** This parameter must be `NULL` when the procedure is invoked from the root of a CDB.
`source_root_name`	The global name of the source root database. In a non-CDB, this parameter must be `NULL`. In a CDB, both `source_database` and `source_root_name` must be specified to identify a specific container.

Usage Notes

The following usage notes apply to this procedure:

- Instantiation SCNs and LCRs
- The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Outbound Servers
- The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Inbound Servers
- The SET_TABLE_INSTANTIATION_SCN Procedure and CDBs

Instantiation SCNs and LCRs

If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at some destination database, then the apply component at the destination database disregards the LCR. Otherwise, the apply component applies the LCR.

The table instantiation SCN specified by this procedure is used on the following types of LCRs:

- Row LCRs for the table
- DDL LCRs that have a non-`NULL` `base_table_owner` and `base_table_name` specified, except for DDL LCRs with a `command_type` of `CREATE TABLE`

For example, the table instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of `ALTER TABLE` or `CREATE TRIGGER`.

Note:

The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:

- SET_GLOBAL_INSTANTIATION_SCN Procedure
- SET_SCHEMA_INSTANTIATION_SCN Procedure
- LCR$_ROW_RECORD Type for more information about row LCRs
- LCR$_DDL_RECORD Type for more information about DDL LCRs

The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Outbound Servers

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.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

See Also:

Oracle Database XStream Guide for more information about outbound servers and instantiation SCNs

The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Inbound Servers

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

The SET_TABLE_INSTANTIATION_SCN Procedure and CDBs

In a CDB, this procedure must be invoked from the same container as the apply process that uses the instantiation SCN information.

SET_UPDATE_CONFLICT_HANDLER Procedure

This procedure adds, modifies, or removes a prebuilt update conflict handler for the specified object.

Syntax

```
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
   object_name           IN  VARCHAR2,
    method_name          IN   VARCHAR2,
    resolution_column    IN   VARCHAR2,
    column_list          IN   DBMS_UTILITY.NAME_ARRAY,
    apply_database_link  IN   VARCHAR2   DEFAULT NULL);
```

Parameters

Table 25-27 SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
`object_name`	The schema and name of the table, specified as `[`*`schema_name`*`.]`*`object_name`*, for which an update conflict handler is being added, modified, or removed. For example, if an update conflict handler is being added for table `employees` owned by user `hr`, then specify `hr.employees`. If the schema is not specified, then the current user is the default.
`method_name`	Type of update conflict handler to create. You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained: • `MAXIMUM`: Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database. • `MINIMUM`: Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database. • `OVERWRITE`: Applies the column list from the source database, overwriting the column values at the destination database. • `DISCARD`: Retains the column list from the destination database, discarding the column list from the source database. If `NULL`, then the procedure removes any existing update conflict handler with the same `object_name`, `resolution_column`, and `column_list`. If non-`NULL`, then the procedure replaces any existing update conflict handler with the same `object_name` and `resolution_column`.
`resolution_column`	Name of the column used to uniquely identify an update conflict handler. For the `MAXIMUM` and `MINIMUM` prebuilt methods, the resolution column is also used to resolve the conflict. The resolution column must be one of the columns listed in the `column_list` parameter. `NULL` is not allowed for this parameter. For the `OVERWRITE` and `DISCARD` prebuilt methods, you can specify any column in the column list.
`column_list`	List of columns for which the conflict handler is called. The same column cannot be in more than one column list. If a conflict occurs for one or more of the columns in the list when an apply component tries to apply a row logical change record (row LCR), then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list. **Note:** Prebuilt update conflict handlers do not support LOB, `LONG`, `LONG RAW`, user-defined type, and Oracle-suppled type columns. Therefore, you should not include these types of columns in the `column_list` parameter.

Table 25-27 (Cont.) SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database. **Note:** Currently, conflict handlers are not supported when applying changes to a non-Oracle database.

Usage Notes

The following usage notes apply to this procedure:

- Modifying an Existing Update Conflict Handler
- Removing an Existing Update Conflict Handler
- Series of Actions for Conflicts
- Procedure DML Handlers for Conflicts
- A Column Can Be in Only One Column List
- Update Conflict Handlers and Non-Oracle Databases
- The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Outbound Servers
- The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Inbound Servers

Modifying an Existing Update Conflict Handler

If you want to modify an existing update conflict handler, then you specify the table and resolution column of an the existing update conflict handler. You can modify the prebuilt method or the column list.

Removing an Existing Update Conflict Handler

If you want to remove an existing update conflict handler, then specify `NULL` for the prebuilt method and specify the table, column list, and resolution column of the existing update conflict handler.

Series of Actions for Conflicts

If an update conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate update conflict handler to resolve the conflict
2. If no update conflict handler is specified or if the update conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply component, table, and operation to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

See Also:

"Signature of a DML Handler Procedure or Error Handler Procedure" for information about setting an error handler

Procedure DML Handlers for Conflicts

If you cannot use a prebuilt update conflict handler to meet your requirements, then you can create a PL/SQL procedure to use as a custom conflict handler. You use the `SET_DML_HANDLER` procedure to designate one or more custom conflict handlers for a particular table. In addition, a custom conflict handler can process LOB columns and use LOB assembly.

See Also:

SET_DML_HANDLER Procedure

A Column Can Be in Only One Column List

When a column is in a column list, and you try to add the same column to another column list, this procedure returns the following error:

```
ORA-00001: UNIQUE CONSTRAINT (SYS.APPLY$_CONF_HDLR_COLUMNS_UNQ1) VIOLATED
```

Update Conflict Handlers and Non-Oracle Databases

Setting an update conflict handler for an apply component that is applying to a non-Oracle database is not supported.

The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

Examples

The following is an example for setting an update conflict handler for the `employees` table in the `hr` schema:

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'salary';
  cols(2) := 'commission_pct';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.employees',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'salary',
    column_list           =>  cols);
END;
/
```

This example sets a conflict handler that is called if a conflict occurs for the `salary` or `commission_pct` column in the `hr.employees` table. If such a conflict occurs, then the `salary` column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the `job_id` column, then this conflict handler is not called.

SET_VALUE_DEPENDENCY Procedure

This procedure sets or removes a value dependency. A value dependency is a virtual dependency definition that defines a relationship between the columns of two or more tables.

An apply component uses the name of a value dependencies to detect dependencies between row logical change records (row LCRs) that contain the columns defined in the value dependency. Value dependencies can define virtual foreign key relationships between tables, but, unlike foreign key relationships, value dependencies can involve more than two database objects.

This procedure is overloaded. The `attribute_list` and `attribute_table` parameters are mutually exclusive.

Syntax

```
DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY(
   dependency_name IN VARCHAR2,
   object_name     IN VARCHAR2,
   attribute_list  IN VARCHAR2);

DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY(
   dependency_name IN VARCHAR2,
   object_name     IN VARCHAR2,
   attribute_table IN DBMS_UTILITY.NAME_ARRAY);
```

Parameters

Table 25-28 SET_VALUE_DEPENDENCY Procedure Parameters

Parameter	Description
`dependency_name`	The name of the value dependency. If a dependency with the specified name does not exist, then it is created. If a dependency with the specified name exists, then the specified object and attributes are added to the dependency. If `NULL`, an error is raised.
`object_name`	The name of the table, specified as `[`*`schema_name`*`.]`*`table_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. If `NULL` and the specified dependency exists, then the dependency is removed. If `NULL` and the specified dependency does not exist, then an error is raised. If `NULL`, then `attribute_list` and `attribute_table` also must be `NULL`.
`attribute_list`	A comma-delimited list of column names in the table. There must be no spaces between entries.
`attribute_table`	A PL/SQL associative array of type `DBMS_UTILITY.NAME_ARRAY` that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be `NULL` terminated.

Usage Notes

The following usage notes apply to this procedure:

- The SET_VALUE_DEPENDENCY Procedure and XStream Outbound Servers
- The SET_VALUE_DEPENDENCY Procedure and XStream Inbound Servers

The SET_VALUE_DEPENDENCY Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The SET_VALUE_DEPENDENCY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

START_APPLY Procedure

This procedure directs the apply component to start applying messages.

Syntax

```
DBMS_APPLY_ADM.START_APPLY(
   apply_name   IN   VARCHAR2);
```

Parameter

Table 25-29 START_APPLY Procedure Parameter

Parameter	Description
`apply_name`	The apply component name. A `NULL` setting is not allowed. Do not specify an owner.

Usage Notes

The following usage notes apply to this procedure:

- Apply Component Status
- The START_APPLY Procedure and XStream Outbound Servers
- The START_APPLY Procedure and XStream Inbound Servers

Apply Component Status

The apply component status is persistently recorded. Hence, if the status is `ENABLED`, then the apply component is started upon database instance startup. An apply component (*`annn`*) is an Oracle background process. The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the start status of an apply component.

The START_APPLY Procedure and XStream Outbound Servers

This procedure functions the same way for apply processes and outbound servers.

The START_APPLY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

STOP_APPLY Procedure

This procedure stops the apply component from applying messages and rolls back any unfinished transactions being applied.

Syntax

```
DBMS_APPLY_ADM.STOP_APPLY(
   apply_name   IN  VARCHAR2,
   force        IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 25-30 STOP_APPLY Procedure Parameters

Parameter	Description
`apply_name`	The apply component name. A `NULL` setting is not allowed. Do not specify an owner.
`force`	If `TRUE`, then the procedure stops the apply component as soon as possible. If `FALSE`, then the procedure stops the apply component after ensuring that there are no gaps in the set of applied transactions. The behavior of the apply component depends on the setting specified for the `force` parameter and the setting specified for the `commit_serialization` apply component parameter. See "Usage Notes" for more information.

Usage Notes

The following usage notes apply to this procedure:

- Apply Component Status
- Queue Subprograms Have No Effect on Apply Component Status
- The STOP_APPLY force Parameter and the commit_serialization Apply Parameter
- The STOP_APPLY Procedure and XStream Outbound Servers
- The STOP_APPLY Procedure and XStream Inbound Servers

Apply Component Status

The apply component status is persistently recorded. Hence, if the status is `DISABLED` or `ABORTED`, then the apply component is not started upon database instance startup.

Queue Subprograms Have No Effect on Apply Component Status

The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the `STOP` status of an apply component.

The STOP_APPLY force Parameter and the commit_serialization Apply Parameter

The following table describes apply component behavior for each setting of the `force` parameter in the `STOP_APPLY` procedure and the `commit_serialization` apply component parameter. In all cases, the apply component rolls back any unfinished transactions when it stops.

force	commit_serialization	Apply Component Behavior
TRUE	FULL	The apply component stops immediately and does not apply any unfinished transactions.
TRUE	DEPENDENT_TRANSACTIONS	When the apply component stops, some transactions that have been applied locally might have committed at the source database at a later point in time than some transactions that have not been applied locally.
FALSE	FULL	The apply component stops after applying the next uncommitted transaction in the commit order, if any such transaction is in progress.
FALSE	DEPENDENT_TRANSACTIONS	Before stopping, the apply component applies all of the transactions that have a commit time that is earlier than the applied transaction with the most recent commit time.

For example, assume that the `commit_serialization` apply component parameter is set to `DEPENDENT_TRANSACTIONS` and there are three transactions: transaction 1 has the earliest commit time, transaction 2 is committed after transaction 1, and transaction 3 has the latest commit time. Also assume that an apply component has applied transaction 1 and transaction 3 and is in the process of applying transaction 2 when the `STOP_APPLY` procedure is run. Given this scenario, if the `force` parameter is set to `TRUE`, then transaction 2 is not applied, and the apply component stops (transaction 2 is rolled back). If, however, the `force` parameter is set to `FALSE`, then transaction 2 is applied before the apply component stops.

A different scenario would result if the `commit_serialization` apply component parameter is set to `FULL`. For example, assume that the `commit_serialization` apply component parameter is set to `FULL` and there are three transactions: transaction A has the earliest commit time, transaction B is committed after transaction A, and transaction C has the latest commit time. In this case, the apply component has applied transaction A and is in the process of applying transactions B and C when the `STOP_APPLY` procedure is run. Given this scenario, if the `force` parameter is set to `TRUE`, then transactions B and C are not applied, and the apply component stops (transactions B and C are rolled back). If, however, the `force` parameter is set to `FALSE`, then transaction B is applied before the apply component stops, and transaction C is rolled back.

See Also:

SET_PARAMETER Procedure for more information about the `commit_serialization` apply component parameter

The STOP_APPLY Procedure and XStream Outbound Servers

This procedure functions the same way for apply processes and outbound servers.

The STOP_APPLY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.