Streams Setup

Author: G S Chapman

Date: 24th October 2008

Version: 1.0

Location of Document:

DOCUMENT HISTORY

Version	Date	Changed By:	Remarks
1.0	24/10/08	G S Chapman	Initial version

DOCUMENT DISTRIBUTION

Copy No	Name	Role	Organisation

DOCUMENT REFERENCES

Document Name	Originator	Part Number	Version	Date
Oracle Streams Performance and Troubleshooting Best Practices: Oracle Database 10g Release 2 MAA white paper.	Oracle Corporation	White Paper		
Oracle Metalink Technical Note (strmmon)	Oracle Corporation	Note: 290605.1		
Oracle Corporation Streams Concepts and Administration Manual	Oracle Corporation	B14229		
Oracle Corporation Streams Replication Administrators Guide	Oracle Corporation	B14228		
Oracle Corporation PL-SQL Packages and Types Reference Manual	Oracle Corporation	B14258		
Oracle Corporation Database Concepts Manual.	Oracle Corporation	B14220		
Oracle Corporation Data Warehousing Guide	Oracle Corporation	B14223		
Oracle Streams Performance and Troubleshooting Best Practices: Oracle Database 10g Release 2 MAA white paper.	Oracle Corporation	White Paper		

TABLE OF CONTENTS

1 STREAMS DESCRIPTION 6
1.1 Streams Data Capture 6
1.2 Logical Change Records (LCRs) 7
1.3 OLS implication 8
1.4 Solution Overview 8
1.5 Change table definition 9
2 CONFIGURATION 10
2.1 Streams Administrator 10
2.2 Streams Initialisation Parameters 10
2.3 Supplementary Logging 12
2.4 Queues 13
2.4.1 Source Database Queue 13
2.4.2 Target Database Queue 13
2.4.3 Queues in a RAC configuration 13
2.5 Propagation 14
2.5.1 Database Link 14
2.5.2 Table Propagation Rule 14
2.5.3 Schema Propagation Rule 14
2.5.4 Propagation settings 15
2.6 Capture Process 16
2.6.1 Table Capture Rule 16
2.7 Default Apply Process 17
2.7.1 Table Apply Rule 17
2.7.2 Schema Apply Rule 18
2.8 Change Table Apply Process 19
2.9 Instantiation 22
2.9.1 Use of network datapump 23
2.9.2 Object instantiation 23
2.9.3 Table instantiation call: 23
2.9.4 Schema instantiation call 24
2.10 Startup 24
2.10.1 Start the Apply 24
2.10.2 Start the Capture 24
2.10.3 Starting order 24
2.11 Post Configuration Tasks 25
2.11.1 Set Capture Checkpoint Retention Time Parameter 25
2.11.2 Set apply parallelism to 4. 25
2.12 Error Handling 25
2.13 Housekeeping 26
2.13.1 Stopping the Replication. 26
2.13.2 Removing Streams replication completely. 26
3 MONITORING 28
3.1 Strmmon Utility 28
3.2 DDL Replication 30
3.3 Propagation 30
3.4 Source Queue Growth 30
3.5 Useful tables and views 30
3.6 Monitoring Scripts 34
3.7 Grid Control Monitoring 35
3.7.1 User Defined Metrics 36
3.8 Streams Commander (OSC) 36
4 AUTOMATING THE SETUP 37
5 ADDITIONAL NOTES: 40
5.1 Database Links 40
5.2 Problems with creating capture process. 40
5.3 Database Recovery Scenarios 41

5.4 Queue Ordering 41
5.5 Heartbeat Table 41
5.5.1 To implement a heartbeat table: 42
5.6 Replicating tables with extra columns 42
5.7 Adding a Global Rule to exclude certain DDL statements 44
5.8 Excluding Columns in Capture 44
A1 Streams Utility Package 46
A2 Generation Scripts 48
A3 Usage of written scripts 50
A4 Update release changes 51
A5 Adding a few tables to the replication. 52
A6 Handling Apply Spill 53
A6.1 Apply Spill (10.2) 53
A6.2 Purging Apply Spill 53
A7 Problems restarting Capture process 55
A8 Known problems when removing Streams configuration 57
A9 Using Stream Tags 58
A9.1 Setting the Tag Values Generated by the Current Session 58
A9.2 Getting the Tag Value for the Current Session 58
A9.3 Example using Tags 58
A10 Example of dropping columns and streams in a single database 60
A11 Tracing Streams 64
A12 Deploying Streams in a High Availability Architecture. 65
A12.1 Procedure to recover from a Streams destination failure 65
A12.2 Perform the following steps to recover a Streams destinations failure. 66
A13 Procedure to recover from a Streams SOURCE FAILURE 70
A13.1 Perform the following steps to recover a Streams source failure. 70

TABLE OF FIGURES

Figure 1 - Information flow in a replicated environment 7

TABLES

Table 1 - Change table additional column specifications 9
Table 2 - Streams Initialisation Parameters 12
Table 3 - Strmmon utility parameters 29
Table 4 - Dynamic Streams Views 32
Table 5 - Streams Monitorinq SQL scripts 34
Table 6 - STRMUTIL_PKG procedures. 47
Table 7 - Generation Scripts 49

PURPOSE OF DOCUMENT

The purpose of this document is to:

- Describe at a detailed level the implementation of streams replication
- Define how the components are integrated
- Detail the setup procedures

It is assumed that the reader of this document understands the principles behind being an Oracle DBA.

1 STREAMS DESCRIPTION

There are a number of requirements to make data available between different systems. The method to perform this task is called replication and can be described as the process of sharing database objects and data between multiple databases. To maintain replicated database objects and data at multiple databases, a change to one of these database objects at a database is shared with the other databases. In this way, the database objects and data are kept synchronized at all of the databases in the replication environment. The database where a change originates is called the master (or source) database, and a database where a change is shared is called a destination database.

There are two specific requirements:

- Replication: Simple copy of data between two systems
- Change Capture: The capture of changes made such that they may be used for another purpose.

The method used will be based upon Oracle Streams technology. A streams capture process will be used upon the source system, which will then be propagated to a destination system. On the destination system, an apply process will be used which will 'replicate' the changes to the base tables, and/or where required create the change tables.

A Streams environment can be configured to replicates changes for an entire source database, certain schemas in the source database, or subsets of certain tables in the source database. It is also possible to configure a Streams environment that maintains DML changes, DDL changes, or both. The database objects configured for replication can also be in multiple tablespaces in the source database.

1.1 Streams Data Capture

Streams Data Capture permits the replication of a DML or DDL change upon Oracle database tables in three steps:

1. A capture process or an application creates one or more logical change records (LCRs) and enqueues them into a queue. An LCR is a message with a specific format that describes a database change. A capture process reformats changes captured from the redo log into LCRs, and applications can construct LCRs. If the change was a data manipulation language (DML) operation, then each LCR encapsulates a row change resulting from the DML operation to a shared table at the source database. If the change was a data definition language (DDL) operation, then an LCR encapsulates the DDL change that was made to a shared database object at a source database.
2. A propagation process moves the staged LCR to another queue, which usually resides in a database that is separate from the database where the LCR was captured. An LCR can be propagated to a number of queues before it arrives at a destination database.
3. At a destination database, an apply process consumes the change by applying the LCR to the shared database object. An apply process can dequeue the LCR and apply it directly, or an apply process can dequeue the LCR and send it to an apply handler. In a Streams replication environment, an apply handler performs customized processing of the LCR and then applies the LCR to the shared database object.

Streams technology also enables the apply process to be modified so that a change data table can be created for a 'few' selected objects and a simple application of all other changes to the destination objects. This is achieved by the use of Rules. A rule is a database object that enables a client to perform an action when an event occurs and a condition is satisfied. Rules are evaluated by

a rules engine, which is a built-in part of Oracle. Each of the following mechanisms is a client of the rules engine:

- Capture process
- Propagation
- Apply process

The behaviour of each of these Streams clients is controlled using rules. A rule set contains a collection of rules, and you can associate a positive and a negative rule set with a Streams client. In a replication environment, a Streams client performs an action if an LCR satisfies its rule sets. In general, a change satisfies the rule sets for a Streams client if no rules in the negative rule set evaluate to true for the LCR, and at least one rule in the positive rule set evaluates to true for the LCR. If a Streams client is associated with both a positive and negative rule set, then the negative rule set is always evaluated first.

Step 1 and Step 3 are required, but Step 2 is optional because, in some cases, an application can enqueue an LCR directly into a queue at a destination database. In addition, in a heterogeneous replication environment in which an Oracle database shares information with a non-Oracle database, an apply process can apply changes directly to a non-Oracle database without propagating LCRs.

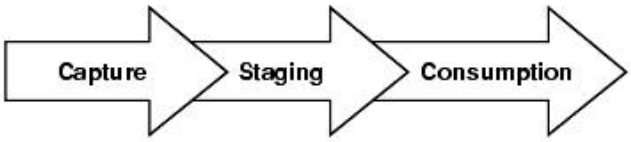

Figure 1 - Information flow in a replicated environment

One advantage that Oracle Streams provides is that it allows one source and multiple destinations. This situation is a Streams environment in which one database is the primary database, and this primary database shares data with several secondary databases. The secondary databases optionally may share data only with the primary database. The secondary databases do not share data directly with each other, but, instead, optionally can share data indirectly with each other through the primary database. This type of environment is sometimes called a "hub and spoke" environment, with the primary database being the hub and the secondary databases being the spokes.

On the destination system a apply process would run, consuming the messages from the queue and in most cases applying it to a destination table. It is possible to have multiple apply processes, for example one might update a ‘copy’ of the source table and the second could be to populate change tables for a selected number of tables.

In the circumstance of the second case it would be the responsibility of the ETL process that would use the change tables to ensure that data integrity is preserved. (Taking account of the SCN numbers on the change records ensuring they are applied in the correct order.)

The Streams information flow:

The Oracle Streams architecture consists of three basic elements or processes as represented in the above diagram:

- Capture
- Staging
- Apply or Consume

1.2 Logical Change Records (LCRs)

The capture process extracts the DML/DDL statements evaluated by the rules engine from the redo log and formats them into events also called Logical

Changed Records. These LCRs will then be placed in the queue or the staging area.

Oracle Streams presents us with control over what information needs to be shared, how it is propagated and to which subscriber (databases). This control can be exercised by tagging each LCR with a tag that identifies in which database the change originated (whether a local database or a remote database, especially in an N-tiered database network), for tracking the information flow or for specifying the set of destination databases that can use the information.

The capture process formats two types of LCRs, the DDL LCR and the row LCR.

The row LCR describes changes made to a single row of a table modified with a single DML statement. Thus, a single DML statement that updates 100 rows in the table will generate a 100 row LCR and a single transaction containing (say) two DML statements, each updating 100 rows will generate 200 LCRs (100 x 2 LCRs).

The DDL LCR, on the other hand, describes the changes in the database objects (such as a DDL statement issued to create, alter or drop a database object).

The following figure shows the Oracle Streams architecture.

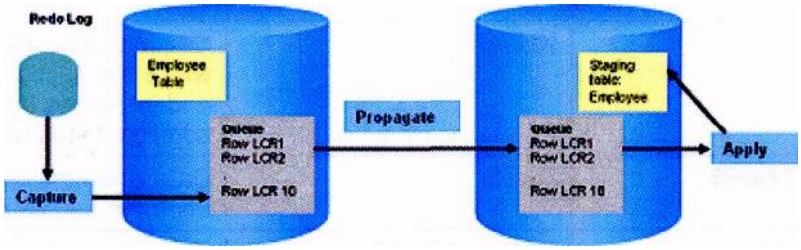

Figure 2 - Streams Architecture

```
/* DML updates only 1 row */
UPDATE employee
SETjob='VP'
WHERE employee_id=11234;
/* DML updates (say) 10 rows
*/
UPDATE employee
SET comm=0.05*salary
WHERE
department='SALES';
```

```
/* DML updates only 1 row */
UPDATE employee
SETjob='VP'
WHERE
employee_id=11234;
/* DML updates (say) 10
rows */
UPDATE employee
SET comm=0.05*salary
WHERE
department='SALES';
```

1.3 OLS implication

The implementation of OLS within the schemas will not impact the replication design. The 'users' who perform the capture and application of the changes are privileged users within the database. These specific user accounts would only be used for the purpose of the 'replication' and for no other purpose.

1.4 Solution Overview

The main objectives are:

- To minimise the impact on the master (source) databases and data fabric.
- To only make destination data changes where data has genuinely changed since the last update.

1.5 Change table definition

The change tables that are populated as a result of the apply procedure defined below needs to be defined as part of the set up of the mechanism. A generic format has been defined which will consist of the existing columns in the source table together with the following fixed columns defined in the table below.

Column Name	Format	Comments
OPERATIONS$	VARCHAR2(2)	Operation being performed, i.e. 1 - Insert, D - Delete, UO - Update old values, UN - Update new values.
CSCN$	NUMBER	System change number on source system.
RSID$	NUMBER	Sequential count for statement with a transaction set. Starts at 1 and increases by 1 for each additional statement.
COMMIT_TIMESTAMP$	DATE	Date when the change was made upon the source database.
SOURCE_SYSTEM$	VARCHAR2(80)	The name of the source system.

Table 1 - Change table additional column specifications

2 CONFIGURATION

These instructions provide details of how to set up and use Streams based replication between databases. The information supplied covers a means of creating a 'Change Data Capture' table upon the destination database, which can be used by ETL processes to record changes made upon the base tables.

2.1 Streams Administrator

It is recommended that a separate user be configured to manage the Streams environment. The standard name used is typically STRMADMIN. This must be a limited access account as it has DBA privileges (note that DBA is only required to support CREATE / ALTER Capture and Apply processes). In addition, it is good practice to create a separate tablespace for the STRMADMIN user (Minimum size 50M). This specific task would be performed by a database administrator.

```
CREATE TABLESPACE STREAMS_TS LOGGING
        EXTENT MANAGEMENT LOCAL AUTOALLOCATE
SEGMENT SPACE MANAGEMENT AUTO
        DATAFILE '+XXXX' SIZE 50M
AUTOEXTEND ON NEXT 50M MAXSIZE UNLIMITED;
```

A STRMADMIN user must be created on each database participating in the Streams environment.

Sample script:

```
CREATE USER strmadmin IDENTIFIED BY password
DEFAULT TABLESPACE streams_ts QUOTA UNLIMITED ON streams_ts;
GRANT dba TO strmadmin;
BEGIN
          dbms_streams_auth.grant_admin_privilege (
     grantee              => 'STRMADMIN',
     grant_privileges     => TRUE);
END;
/
```

The strmadmin user will also need to be supplied with select, insert, update and delete permission upon the tables to be 'replicated' upon the destination system.

The strmadmin user will also require FULL OLS privileges to write to security protected tables IF OLS is implemented upon the target database.

2.2 Streams Initialisation Parameters

A number of database initialisation parameters must be set to ensure Streams works correctly. These are documented in The Streams Administration Guide and in the table below, but must include:

GLOBAL_NAMES = TRUE
JOB_QUEUE_PROCESSES = 4 (or more)
STREAMS_POOL_SIZE = 256M (set even if using
Automatic Shared Memory Management to specify a minimum size)

The following table is extracted from the "Oracle Streams Performance and Troubleshooting Best Practices: Oracle Database 10g Release 2" MAA white paper.

Parameter Name	Description

Parameter Name	Description
AQ_TM_PROCESSES	Do not explicitly set this parameter to 0 or 10. Doing so could disable the queue monitoring processing and impact the Streams pool memory utilization. If this parameter has not been set, the parameter will be auto tuned. If this parameter has been explicitly set to 0 or 10, either in the initialization parameter file or by the ALTER SYSTEM statement, then reset the parameter to 1.
DB_NAME	On each Streams database, specify the name that was given to the database when it was first created. For example, if your source database is called STREAMSS and your target database is called STREAMSD, be sure to set the DB_NAME parameter on each database accordingly. To determine the value of DB_NAME, log into SQL*Plus as SYSDBA. Issue the following query to determine the name: SELECT NAME FROM V$DATABASE;
DB_DOMAIN	On each Streams database, specify the network domain where the database resides. For example, if the source database is in mydomainA.com domain and the target database is mydomainB.com domain, then ensure that the DB_DOMAIN parameter is set to the correct domain. For example, on the source database, specify DB_DOMAIN=mydomainA.com and on the target database, specify DB_DOMAIN=mydomainB.com.
GLOBAL_NAME=TRUE	On each Streams database, specify GLOBAL_NAMES=TRUE to ensure that each database link used by Streams has the same name as the destination's database GLOBAL_NAME to which the link points.
COMPATIBLE=10.2	On each Streams database, set the COMPATIBLE parameter to "10.2" to use the new features available with Oracle Database 10g release 2 (10.2). Note that once you set COMPATIBLE to 10.2, you can no longer downgrade to your previous release.
JOB_QUEUE_PROCESSES=4	On each Streams database, specify JOB_QUEUE_PROCESSES=4 (the minimum recommended value) to start and use four job queue processes *(]nnn)* for Streams propagation jobs. If this parameter is already set, then increment the JOB_QUEUE_PROCESSES parameter by the number of independent Stream propagations.
JOB_QUEUE_INTERVAL=1	On each Streams database, specify how many seconds the job queue is scanned for work. It is recommended that you set this parameter to 1 (seconds) to improve the propagation job performance and minimize the delay for how often propagation jobs will execute.
TIMED_STATISTICS=TRUE	On each Streams database, set the TIMED_STATISTICS=TRUE parameter to allow performance statistics to be gathered for analysis of potential performance bottlenecks. Setting this parameter may have some performance impact. However, it is necessary to collect the statistics required to conduct performance tuning

Parameter Name	Description
	once Streams is running.
STATISTICS_LEVEL=TYPICAL	On each Streams database, set this parameter to TYPICAL to collect the necessary statistics. This parameter works with the TIMED_STATISTICS parameter to determine where performance issues occur when Streams is running. Set both the TIMED_STATISTICS and STATISTICS_LEVEL parameters because: The performance impact is minimal for both. The Automatic Workload Repository (AWR) report and Active Session History (ASH) reports depend on these parameters. Note: Ensure that AWR is installed and running on each Streams database.
SHARED_POOL_SIZE=256M	On each Streams database, set this parameter to a minimum value of 256 MB because Streams uses a significant amount of PL/SQL and library cache components. Future performance analysis may provide recommendations for how much to increase the size of SHARED_POOL_SIZE, especially for the database running Stream apply processes. Also, see the "Oracle Streams Performance and Troubleshooting Best Practices: Oracle Database 10g Release 2" MAA white paper that provides practical tuning and troubleshooting techniques for Streams configurations.
STREAMS_POOL_SIZE=256M	On each database where there is either a capture or apply process running, set the STREAMS_POOL_SIZE parameter to a minimum of 256M. Memory from the Streams pool is used by both the Streams processes and the buffered queue, with the LCRs in the buffered queue the largest component. Use the V$STREAMS_POOL_ADVICE view to help you size the buffered queue. For downstream capture, do not set the STREAMS_POOL_SIZE parameter on the source database if there are no capture processes configured. For local capture, set the STREAMS_POOL_SIZE parameter on the source and target databases.

Table 2 - Streams Initialisation Parameters

2.3 Supplementary Logging

Data Capture works through mining the redo logs on the source database, but for this to work best the source database has to be instructed to perform supplemental logging on the tables to be tracked. This may be set at the database level or upon the individual tables. The following show the command to set the supplemental logging upon the database.

```
alter table emp add supplemental log group log_group_emp(empno)
always;
```

Note: In Oracle 10g release 2, if the STREAMS_ADM package is used to set up the replication as is the situation in these instructions then the supplementary

logging is automatically performed and there is no need to change all the tables individually as a separate step.

It is noted that when the STREAMS_ADM package is used to generate the capture process that it performs an 'ALTER' upon the database to add supplementary logging. This is by design but might not be the expected behaviour.

2.4 Queues

Streams uses buffer queues with a payload type of ANYDATA to propagate messages from the source database to the target database. When a capture process or apply process is created, the process is associated with a specific ANYDATA queue. When a propagation process is created, it is associated with a specific source queue and destination queue, therefore these queues need to be created first. Streams uses buffer (in-memory) secure queues; these secure queues ensure that no other users apart from the STRMADMIN user can access messages in the queue.

2.4.1 Source Database Queue

```
BEGIN
  dbms_streams_adm.set_up_queue(
          queue_table        => 'SRC_QT_1',
          queue_name         => 'SRC_Q1',
          queue_user         => 'STRMADMIN');
END;
/
```

2.4.2 Target Database Queue

```
BEGIN
  dbms_streams_adm.set_up_queue(
          queue_table        => 'TGT_QT_1',
          queue_name         => 'TGT_Q1',
          queue_user         => 'STRMADMIN');
END;
/
```

2.4.3 Queues in a RAC configuration

When Streams is configured in a RAC environment, each queue table has an "owning" instance. All queues within an individual queue table are owned by the same instance. The Streams components (capture/propagation/apply) all use that same owning instance to perform their work. This means that:

- A capture process is run at the owning instance of the source queue.
- A propagation job must run at the owning instance of the queue.
- A propagation job must connect to the owning instance of the target queue.

Ownership of the queue can be configured to remain on a specific instance, as long as that instance is available, by setting the PRIMARY_INSTANCE and/or SECONDARY_INSTANCE parameters of DBMS_AQADM.ALTER_QUEUE_TABLE. If the PRIMARY_INSTANCE is set to a specific instance (i.e., not 0), the queue ownership will return to the specified instance whenever the instance is up.

Capture will automatically follow the ownership of the queue. If the ownership changes while capture is running, capture will stop on the current instance and restart at the new owner instance.

For queues created with Oracle Database 10g Release 2, a service will be created with the service name = schema.queue and the network name SYS$schema.queue.global_name for that queue. If the global_name of the

database does not match the db_name.db_domain name of the database, be sure to include the global_name as a service name in the init.ora.

For propagations created with the Oracle Database 10g Release 2 code with the queue_to_queue parameter to TRUE, the propagation job will deliver only to the specific queue identified. Also, the source dblink for the target database connect descriptor must specify the correct service (global name of the target database) to connect to the target database. For example, the tnsnames.ora entry for the target database should include the CONNECT_DATA clause in the connect descriptor for the target database. This clause should specify (CONNECT_DATA=(SERVICE_NAME='global_name of target database')). Do NOT include a specific INSTANCE in the CONNECT_DATA clause.

2.5 Propagation

2.5.1 Database Link

The propagation process moves the changes to the replicated tables from the source database to the target database in the form of a Logical Change Record (LCR). Therefore, a database link is required to support this. [See 7.1 for further information on the database link setup.] An example is shown below:

CREATE DATABASE LINK TGT CONNECT TO strmadmin IDENTIFIED BY *password using* 'TGT';

Check that the link works.

```
select * from dual@TGT;
```

Note that because the database has the global_names=true setup, the name of the database link must match the name of the target database exactly.

Propagation can be set up for an individual table, an entire schema or the entire database. The choice of which to use will be driven by to specific number of tables and/or schemas to be replicated. An example is given below for both an individual table and also for an entire schema.

2.5.2 Table Propagation Rule

Propagation is set up by adding a rule for each table to be replicated as follows. The first call to the procedure for a streams_name (i.e. the name of the propagation) configures the streams environment appropriately. Propagation is a queue to queue process (i.e. source_queue_name to destination_queue_name)

```
BEGIN
  dbms_streams_adm.add_table_propagation_rules(
        table_name                => 'SCOTT.BMP',
        streams_name              => 'SRC_TO_TGT',
        source_queue_name         => 'STRMADMIN.SRC_Q1',
        destination_queue_name    => 'STRMADMIN.TGT_Q1@TGT',
        include_dml               => TRUE,
        include_ddl=> FALSE,
        source database     => 'SRC',
        inclusion_rule      => TRUE,
        queue_to_queue            => TRUE);
END;
/
```

Multiple destinations can be supported from a single source queue if required.

2.5.3 Schema Propagation Rule

An entire schema can be propagated using the following example.

```
BEGIN
```

```
   DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES (
      schema_name =>   'SCOTT',
       streams_name            =>   ' SRC_TO_TGT ' ,
       source_queue_name =>   ' STRMADMIN. SRC_Q1 ' ,
       destination_queue_name  =>   ' STRMADMIN. TGT_Q1@TGT ',
        include_dml   =>  TRUE,
       include_ddl     =>   FALSE,
        include_tagged_lcr   =>   FALSE,
       source_database      =>    'SRC',
       inclusion_rule =>  TRUE,
      queue_to_queue   =>  TRUE) ;
END;
/
```

Running this procedure performs the following actions:

- Creates a propagation named SRC_TO_TGT. The propagation is created only if it does not already exist.
- Specifies that the propagation propagates LCRs from STRMADMIN.SRC_Q1 in the current database to STRMADMIN.TGT_Q1 in the TGT database.
- Specifies that the propagation uses the TGT database link to propagate the LCRs, because the destination_queue_name parameter contains @TGT.
- Creates a positive rule set and associates it with the propagation, if the propagation does not have a positive rule set, because the inclusion_rule parameter is set to true. The rule set uses the evaluation context SYS.STREAMS$_EVALUATION_CONTEXT. The rule set name is system generated.
- Creates two rules. One rule evaluates to TRUE for row LCRs that contain the results of DML changes to the tables in the hr schema, and the other rule evaluates to FALSE for DDL LCRs that contain DDL changes to the hr schema or to the database objects in the hr schema. The rule names are system generated.
- Adds the two rules to the positive rule set associated with the propagation. The rules are added to the positive rule set because the inclusion_rule parameter is set to true.
- Specifies that the propagation propagates an LCR only if it has a NULL tag, because the include_tagged_lcr parameter is set to false. This behaviour is accomplished through the system-created rules for the propagation.
- Specifies that the source database for the LCRs being propagated is repl .net, which might or might not be the current database. This propagation does not propagate LCRs in the source queue that has a different source database.
- Creates a propagation job for the queue-to-queue propagation.

2.5.4 Propagation settings

There is one propagation parameter that it will probably be desirable to set. This parameter is the LATENCY.

LATENCY=5 Default: 60

It is the maximum wait, in seconds, in the propagation window for a message to be propagated. The default value is 60. Caution needs to be exercised if latency is not specified for this call; otherwise latency will over-write any existing value with the default value.

For example: if the latency is 60 seconds, then during the propagation window, if there are no messages to be propagated messages from that queue for the destination will not be propagated for at least 60 more seconds. It will be at least 60 seconds before the queue will be checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue will not be checked for 10 minutes and if the latency

is 0, then a job queue process will be waiting for messages to be enqueued for the destination and as soon as a message is enqueued it will be propagated.

Propagation parameters can be set using the ALTER_PROPAGATION_SCHEDULE procedure from the DBMS_AQADM package. For example, to set the latency parameter of the streams propagation from the STREAMS_QUEUE owned by STRMADMIN to the target database whose global_name is DEST_DB for the queue Q1, use the following syntax while logged in *as* the Streams Administrator:

```
dbms_aqadm.alter_propagation_schedule(
    queue_name              => ‘’STRMADMIN.SRC_Q1',
    destination              => 'TGT',
    destination_queue       =>'STRMADMIN.TGT_Q1',
    latency                 => 5);
```

2.6 Capture Process

A capture process must be configured for each database from which changes are to be captured. Capture rules can be created for individual tables, an entire schema or the whole database.

If only a sub set of all the tables in a schema will be required and that it is preferable to have specific rules for each table required, rather than a rule for a schema and a 'negative' rule for each table that is not required.

2.6.1 Table Capture Rule

As per propagation, a rule is required for each table and the first call performs the configuration:

```
BEGIN
  dbms_streams_adm. add_table_rules (
     table_name     => ' SCOTT. EMP ' ,
    streams_type    => 'CAPTURE',
    streams_name    => ' SRC_CAPTURE' ,
     queue_name     => ' STRMADMIN . SRC_Q1 ' ,
    include_dml     => TRUE,
    include_ddl     => FALSE,
    inclusion_rule  => TRUE) ;
END;
/
```

It may be found that when this above is run that the session will freeze. This is possibly due to there being active transactions running upon the system, such that it is not possible to perform the alter database to implement supplementary logging. Viewing the 'gv$transaction' table will show if there are any active transactions. All one can do is either cancel the running transaction, which may not be advisable, or to wait until the transactions finish.

Note that the first time a capture rule is created that the system will take a few minutes to create all of the underlying objects. Subsequent capture rules run very quickly.

Do not start the capture processes just created. Oracle recommends that there is only one capture process for each source database.

When a procedure is used in the DBMS_STREAMS_ADM package to add the capture process rules, it automatically runs the PREPARE_TABLE_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, or PREPARE_GLOBAL_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package for the specified table, specified schema, or entire database, respectively, if the capture process is a local capture process or a downstream capture process with a database link to the source database.

If there are many tables to perform a capture process upon, the suggested solution is to have a small SQL procedure to generate the appropriate number of 'add_table_rule' commands within a script and then execute the script. This will be easier to manage that to have to run a number of 'delete_rule' commands.

The appropriate procedure must be run to prepare for instantiation manually if any of the following conditions is true:

- The DBMS_RULE_ADM package is used to add or modify rules.
- An existing capture process is used and there are no added capture process rules for any shared object.
- A downstream capture process is used with no database link to the source database.

See section 2.11.1 for an additional post setup parameter to be modified.

2.7 Default Apply Process

A default apply process must be configured for each database to which changes are to be applied (i.e. replicated). As with the other mechanisms such as propagation and capture, the apply process may be defined at a table, schema or database level.

2.7.1 Table Apply Rule

As per propagation, a rule is required for each table and the first call performs the configuration:

```
BEGIN
  dbms_streams_adm. add_table_rules (
     table_name       =>       'SCOTT. EMP',
     streams_type     =>       'APPLY',
     streams_name     =>       'TGT_APPLY',
     queue_name       =>       'STRMADMIN . TGT_Q1',
     include_dml      =>       TRUE,
     include_ddl      =>       FALSE,
     source_database            =>       'SRC',
     inclusion_rule   =>       TRUE) ;
END;
/
```

One common request is to convert the source schema to a different target schema; this can easily be achieved by using a transformation. The use of the transformation is shown in the following script which would be used **instead** of the above.

```
DECLARE
  l_dml_rule_name_all_rules . rule_name%TYPE;
  l_ddl_rule_name_all_rules . rule_name%TYPE;
BEGIN
  dbms_streams_adm. add_table_rules (
    table_name => 'SCOTT. EMP',
     streams_type => 'APPLY',
     streams_name => 'TGT_APPLY',
    queue_name     => 'STRMADMIN. TGT_Q1',
     include_dml    => TRUE,
     include_ddl    => FALSE,
     source_database       => 'SRC',
     inclusion_rule  => TRUE,
     dml_rule_name         => l_dml_rule_name,
     ddl_rule_name         => l_ddl_rule_name) ;

  dbms_streams_adm. rename_schema (
     rule_name   => l_dml_rule_name, from_schema_name  => 'SCOTT',
    to_schema_name     => 'STAGING' );
```

```
END;
/
```

Do not start the apply processes just created.

Optionally add parameters to the apply process. For example, to ensure that the apply process stops when an error is encountered:

```
BEGIN
  dbms_apply_adm. set_parameter (
     apply_name => 'TGT_APPLY',
    parameter  => ' DISABLE_ON_ERROR',
     value     => 'Y');
END;
/
```

See also section 4. 11. 1.2 for an additional step to perform on the apply process.

2.7.2 Schema Apply Rule

If all of the tables are required in a new schema it would be possible to use a schema rule rather than individual table rules as described above. This could then be altered for any tables for which no 'replicated' copy is required and only a change table is needed. This would be a 'negative' rule. At the current time the requirements of the different destination systems are unknown so it is not possible to be certain of the final usage.

```
DECLARE
  l_dml_rule_name_all_rules.rule_name%TYPE;
  l_ddl_rule_name_all_rules.rule_name%TYPE;
BEGIN
  dbms_streams_adm.add_schema_rules(
    schema name => 'SCOTT',
    streams_type => 'APPLY',
    streams_name => 'TGT_APPLY',
    queue_name   => 'STMADMIN.TGT_Q1',
    include_dml => TRUE,
    include_ddl => FALSE,
    source_database => 'SRC',
    inclusion_rule => TRUE,
    dml_rule_name => l_dml_rule_name,
    ddl_rule_name => l_ddl_rule_name);

  dbms_streams_adm.rename_schema (
    rule_name => l_dml_rule_name,
   from_schema_name => 'SCOTT',
   to_schema_name => 'STAGING');

END;
/
```

An example of a negative rule set might be as follows:

```
BEGIN
--
  dbms_streams_adm.add_table_rules(
```

```
        table_name => 'SCOTT.EMP',
        streams_type => 'APPLY',
        streams_name => 'TGT_APPLY',
        queue_name => 'STRMADMIN.TGT_Q1',
        include_dml => TRUE,
        include_ddl => FALSE,
        source_database => 'SRC',
        inclusion_rule => FALSE);
END;
/
```

2.8 Change Table Apply Process

There are a number of tables that require a change table to be maintained (as well as being processed by the default apply process). This can be achieved by adding another apply process which in effect adds another subscriber to the destination queue (multi-subscriber), so the same message is processed twice (i.e. we do not need to capture the change twice).

The first step is to create an apply handler procedure, which reads and manipulates the LCR (including changing the schema name to STAGING and the target table to include a suffix of _CT in this example):

```
create or replace PACKAGE cdc_handler_pkg
AS
  PROCEDURE prc_insert_ct (pi_lcr_anydata IN SYS.ANYDATA);
    l_count NUMBER := 0;
    l_cscn  NUMBER := 0;
END cdc_handler_pkg;
/

create or replace PACKAGE BODY cdc_handler_pkg
AS
--
  PROCEDURE prc_insert_ct(pi_lcr_anydata IN SYS.ANYDATA)
  AS
  --
    l_lcr     SYS.LCR$_ROW_RECORD;
    l_new_lcr         SYS.LCR$_ROW_LIST;
    l_old_lcr         SYS.LCR$_ROW_LIST;
    l_rc      PLS_INTEGER;
    l_operation       VARCHAR(2) := 'I';
  l_object_owner      VARCHAR2(30);
  l_object_name       VARCHAR2(30);
  BEGIN
  — Get the LCR details and change target schema owner and table name
   l_rc         := pi_lcr_anydata.GetObject (l_lcr);
   l_operation          := UPPER (SUBSTR(l_lcr.get_command_type(),1,1));
   l_object_owner       := l_lcr.get_object_owner;
   l_object_name        := l_lcr.get_object_name;
    -- Comment out the following line if schema changed in a transformation call
  l_lcr.set_object_owner ( 'STAGING' );
   l_lcr.set_object_name (l_object_name || '_CT');
   IF l_cscn <> l_lcr.get_commit_scn
   THEN
      l_count := 1;
     l_cscn  := l_lcr.get_commit_scn; ELSE
       l_count : = l_count + 1 ;
   END IF;
    — Convert UPDATE / DELETE to INSERT
   IF l_operation IN ('U','D') THEN
     IF l_operation = 'U1 THEN
```

```
        l_old_lcr := l_lcr.get_values ( 'old' ) ;
        l_new_lcr := l_lcr.get_values ( 'new' ) ;
         l_operation := 'UN'; ELSE
        l_new_lcr := l_lcr.get_values ( 'old');
       END IF;
        —- Set appropriate values and delete old values as INSERT LCR cannot have them
      l_lcr.set_values ( 'new', l_new_lcr);
      l_lcr.set_command_type ('INSERT'); l_lcr.set_values( 'old', NULL);
     END IF;
      -— Add new columns to LCR and execute
    l_lcr.add_column ( 'new', 'OPERATION$' , anydata.convertvarchar2 (l_operation));
    l_lcr.add_column ( 'new', 'CSCN$' , anydata.convertnumber (l_lcr.get_commit_scn));
    l_lcr.add_column ( 'new', 'RSID$' , anydata.convertnumber (l_count));
    l_lcr.add_column( 'new' , 'COMMIT_TIMESTAMP$',
       anydata.convertdate (l_lcr.get_source_time));
    l_lcr.add_column ( 'new' , 'SOURCE_SYSTEM$',
       anydata.convertvarchar2 (l_lcr.get_source_database_name));
    l_lcr.execute (true);
     IF l_operation = 'UN' THEN
      l_lcr.set_values ( 'new',l_old_lcr);
      l_lcr.add_column ( •new', 'OPERATION$', anydata.convertvarchar2 ('UO'));
      l_lcr.add_column ( 'new' , 'CSCN$',
         anydata.convertnumber (l_lcr.get_commit_scn));
      l_lcr.add_column( 'new' , 'RSID$ ' , anydata.convertnumber (l_count));
      l_lcr.add_column( •new1 , 'COMMIT_TIMESTAMP$ ' ,
         anydata.convertdate (l_lcr.get_source_time));
       l_lcr.add_column ( ' new ' , ' SOORCE_SYSTEM$ ' ,
         anydata. convertvarchar2 (l_lcr.get_source_database_name));
       1_lcr.execute (true);
     END IF;

  END prc_insert_ct;
END cdc_handler_pkg;
/
show err
```

Note: The example code above has been formatted to display within this document and may not accurately reflect the format of the actual code used.

The apply process example above has been made specifically generic to make it applicable to multiple (all) tables for which change capture is required. Additional information to be captured may include change time etc. The procedure above also illustrates that 'updates' also capture the 'old' values as well as the 'new' values. This required the creation of a second LCR for the 'updates' to place data into the change tables.

Run the following to configure the apply process:

```
BEGIN

   dbms streams adm . add table rules (
      table name         => 'SCOTT.EMP',
      streams type       => 'APPLY',
      streams name       => 'TGT CDC APPLY',
      queue name         => 'STRMADMIN.TGT Q1',
      include dml        => TRUE,
      include ddl        => FALSE,
      source database    => 'SRC',
      inclusion_rule     => TRUE);

END;

/
```

The procedure created above must now be set as the DML handler for the table. Note that a separate call must be made for each DML operation.

```
BEGIN
  dbms_apply_adm.set_dml_handler(
    object_name        => 'SCOTT.EMP',
    object_type        => 'TABLE',
    operation_name     => 'INSERT',
    error_handler      => FALSE,
    user_procedure     => 'CDC_HANDLER_PKG.PRC_INSERT_CT'
   apply_database_link  => NULL,
    apply_name   ~     => 'TGT_CDC_APPLY');
  dbms_apply_adm.set_dml_handler(
    object_name        => 'SCOTT.EMP',
    object_type        => 'TABLE',
    operation_name     => 'UPDATE',
    error_handler      => FALSE,
   user_procedure      => 'CDC_HANDLER_PKG.PRC_INSERT_CT'
    apply_database_link => NULL,
  apply_name            => 'TGT_CDC_APPLY');
dbms_apply_adm.set_dml_handler(
   object_name     => 'SCOTT.EMP',
    object_type        => 'TABLE',
    operation_name     => 'DELETE',
    error_handler      => FALSE,
   user_procedure     => 'CDC_HANDLER_PKG.PRC_INSERT_CT'
    apply_database_link => NULL,
   apply_name       => 'TGT_CDC_APPLY');
END;
/
```

Optionally add parameters to the apply process. For example, to ensure that the apply process stops when an error is encountered:

```
BEGIN
  dbms_apply_adm.set_parameter(
    apply_name => 'TGT_CDC_APPLY',
    parameter => 'DISABLE_ON_ERROR',
     value     => 'Y');
END;
/
```

An alternative method to changing the schema name would be to change it in the transformation process as illustrated for the normal apply process described earlier. There are a few extra changes required in the scripts for the CDC_Apply scripts.

Firstly the package procedure needs to have the following line (10) removed or commented out.

```
    l_lcr.set_object_owner ('STAGING');
```

The table rules scripts will therefore become as follows instead of the above.

```
DECLARE
  l_dml_rule_name_all_rules.rule_name%TYPE;
  l_ddl_rule_name_all_rules.rule_name%TYPE;
BEGIN
    dbms_streams_adm.add_table_rules (
            table_name      => 'SCOTT.EMP',
            streams_type    => 'APPLY',
            streams_name    => 'TGT_CDC_APPLY',
```

```
            queue_name      => 'STRMADMIN.TGT_Q1',
            include_dml     => TRUE,
            include_ddl     => FALSE,
            source_database         => 'SRC',
            Inclusion_rule  => TRUE,
            dml_rule_name => l_dml_rule_name,
            ddl_rule_name  => l_ddl_rule_name);

    dbms_streams_adm.rename_schema (
            rule_name   => l_dml_rule_name,
            from_schema_name  => 'SCOTT',
            to_schema_name     => 'STAGING');

    dbms_apply_adm.set_dml_handler(
            object_name        => 'STAGING.EMP',
            object_type      => 'TABLE',
            operation_name     => 'INSERT',
            error_handler      => FALSE,
            user_procedure     => 'CDC_HANDLER_PKG.PRC_INSERT_CT'
            apply_database_link => NULL,
            apply_name => 'TGT_CDC_APPLY');

    dbms_apply_adm.set_dml_handler(
            object_name        => 'STAGING.EMP',
            object_type        => 'TABLE',
            operation_name     => 'UPDATE',
            error_handler      => FALSE,
            user_procedure     => 'CDC_HANDLER_PKG.PRC_INSERT_CT'
            apply_database_link => NULL,
            apply_name         => 'TGT_CDC_APPLY');

    dbms_apply_adm.set_dml_handler(
            object_name        => 'STAGING.EMP',
            object_type     => 'TABLE',
            operation_name     => 'DELETE',
            error_handler      => FALSE,
            user_procedure     => 'CDC_HANDLER_PKG.PRC_INSERT_CT'
            apply_database_link => NULL,
            apply_name         => 'TGT CDC APPLY');
END;
/
```

2.9 Instantiation

Instantiation is the setting of the SCN for each database object for which changes are applied by an apply process. If the database objects do not exist at *a* destination database, then instantiate them using export/import, transportable tablespaces, or RMAN. If the database objects already exist at a destination database, then set the instantiation SCNs for them manually.

To instantiate database objects using export/import, first export them at the source database. Next, import them at the destination database.

If the original Export utility is used, then set the OBJECT_CONSISTENT export parameter to y. Regardless of whether you use Data Pump export or original export, it is possible to specify a more stringent degree of consistency by using an export parameter such as FLASHBACK_SCN or FLASHBACK_TIME.

If the original Import utility is used, then set the STREAMS_INSTANTIATION import parameter to y.

2.9.1 Use of network datapump

One common method of ensuring that the destination database has the objects at the same state as the source database is to create the target object using the Oracle supplied utility 'impdp'.

For this a script is run at the target database to pull the database objects over the network onto the target database. No dump file is created on the operating system using this method. The use of a parameter file is often used to make it easier to use the impdp command. Such a parameter file might look as follows:

```
REMAP_SCHEMA=SCOTT:STAGING
SCHEMAS=SCOTT
LOGFILE=imp_scott.log
NETWORK_LINK=SRC
DIRECTORY=DATAPUMPDIR
```

The network link is a database link name defined to the database directed at the source database. The logfile is a record of the script as it runs created in the database directory pointed to by the DIRECTORY parameter. The schema name required is obvious and the remap_schema statement enables the objects on the target database to belong to a different owner from those of the source.

To invoke the script the following command is issued:

```
impdp user/password parfile=parfile
```

If the password is not specified the user is prompted for the value when the command is issued. There are a number of other optional parameters and the interested reader is referred to the official Oracle documentation.

2.9.2 Object instantiation

To set the instantiation SCN for a table, schema, or database manually, run the appropriate procedure or procedures in the DBMS_APPLY_ADM package at the destination database:

```
SET_TABLE_INSTANTIATION_SCN
SET_SCHEMA_INSTANTIATION_SCN
SET_GLOBAL_INSTANTIATION_SCN
```

When any one of these procedures is run, it is important to ensure that the shared objects at the destination database are consistent with the source database as of the instantiation SCN.

If the SET_GLOBAL_INSTANTIATION_SCN is run at a destination database, then set the recursive parameter for this procedure to true so that the instantiation SCN also is set for each schema at the destination database and for the tables owned by these schemas.

If the SET_SCHEMA_INSTANTIATION_SCN is run at a destination database, then set the recursive parameter for this procedure to true so that the instantiation SCN also is set for each table in the schema.

If the recursive parameter to set to true in the SET_GLOBAL_INSTANTIATION_SCN procedure, or within the SET_SCHEMA_INSTANTIATION_SCN procedure; 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 user who executes the procedure.

2.9.3 Table instantiation call:

```
DECLARE
  vlscn NUMBER;
BEGIN
  vlscn := dbms_flashback.get_system_change_number();
  dbms_apply_adm.set_table_instantiation_scn@TGT(
    source_object_name   => 'SCOTT.BMP',
    source_database_name => 'SRC',
```

```
    instantiation sen   => vlscn};
END;
/
```

2.9.4 Schema instantiation call

```
DECLARE
  vlscn NUMBER;
BEGIN
  vlscn := dbms_flashback. get_system_change_number ();
  dbms_apply_adm.set_table_instantiation_scn@TGT (
    source_schema_name   => 'SCOTT.EMP',
    source_database_name => 'SRC',
    recursive      => TRUE,
    instantiation_scn    => vlscn);
END;
/
```

Alternatively, it is possible to perform a metadata export/import to set the instantiation SCNs for existing database objects. If this option is chosen, then make sure no rows are imported. Also, make sure the shared objects at all of the destination databases are consistent with the source database that performed the export at the time of the export. If there is sharing of DML changes only, then table level export/import is sufficient. If there is sharing of DDL changes also, then additional considerations apply.

In the suggested solution it is important to note that changes to the tables upon the destination database should be used for read only purposes, otherwise there is a high risk that 'replicated' changes from the source database may encounter errors. This is a configuration option.

2.10 Startup

As the capture and apply processes define above are not automatically started on creation, they must be started manually. (Note: apply before capture).

Note that the propagation processes are started on creation.

2.10.1 Start the Apply

```
BEGIN
  dbms_apply_adm.start_apply (apply_name => 'TGT_APPLY');
END;
/

BEGIN
  dbms_apply_adm.start_apply (
        apply_name => 'TGT_CDC_APPLY');
END;
 /
```

2.10.2 Start the Capture

```
BEGIN
  dbms_capture_adm.start_capture (
    capture_name => 'SRC_CAPTURE'
END;
/
```

2.10.3 Starting order

The capture process must be created before the relevant objects are instantiated at a remote destination database.

The propagations and apply processes must be created before starting the capture process, and the objects must be instantiated before running the whole stream.

If the propagation process has been stopped, possibly for maintenance purposes, it can be restarted again by running the following procedure.

```
BEGIN
  dbms_propagation_adm.start_propagation(
        propagation_name => 'SRC_TO_TGT' );
END;
/
```

2.11 Post Configuration Tasks

After setting up your downstream or local capture configuration, perform the following steps to finalize and verify the configuration:

2.11.1 Set Capture Checkpoint Retention Time Parameter

Set the CHECKPOINT_RETENTION_TIME capture parameter to specify the number of days of checkpoints the capture process will retain. The default value for this parameter is 60 days but the recommended initial setting is 7 days.

Changing this parameter to 7 days can reduce the volume of checkpoint information that the capture process purges, and therefore improves the overall performance of the capture process.

For example:

```
BEGIN
    DBMS_CAPTURE_ADM.ALTER_CAPTURE(
        capture_name    => 'SRC_CAPTURE',
        checkpoint_retention_time => 7 );
END;
/
```

Query the CAPTURE_NAME of the DBA_CAPTURE view to obtain the name of the capture process.

2.11.2 Set apply parallelism to 4.

As a starting point, set the degree of parallelism for the apply process to "4" as shown in the following example:

```
BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        'TGT_APPLY' , 'PARALLELISM', '4');
END;
/
```

See the *Oracle Streams Performance and Troubleshooting Best Practices* [Table 1] white paper for a discussion about how to determine the optimum degree of parallelism.

Note: The example show above for this step specifies the name TGT_APPLY for the apply process. To find the name of the apply process in the actual configuration; query the APPLY_NAME column of the DBA_APPLY view on the target database.

2.12 Error Handling

If the apply process encounters an unhandled error when it tries to apply an LCR at the destination Oracle database, then the transaction containing the LCR is placed in an exception queue in the Oracle database that is running the apply process. The apply process detects data conflicts and uses automatic conflict resolution. Any data conflicts encountered that can not be resolved are treated as apply errors.

All errors should be investigated to determine the cause of the error. Once identified the particular apply change can either be removed or re-processed as appropriate.

2.13 Housekeeping

2.13.1 Stopping the Replication.

To stop the replication the following steps must be performed in the following order. First stop the capture process.

Stop the Capture

```
BEGIN
  dbms_capture_adm.stop_capture (
        capture_name => 'SRC_CAPTURE');
END;
/
```

Once the capture has stopped ensure that all captured changes have been propagated to the destination system before stopping the propagation process.

Stop the propagation

```
BEGIN
  dbms_propagation_adm.stop_propagation (
        propagation_name => 'SRC_TO_TGT');
END;
/
```

Now the apply processes for the CDC capture and the simple replication can be stopped.

Stop the Apply

```
BEGIN
  dbms_apply_adm. stop_apply (apply_name => 'TGT_APPLY');
END;
/

BEGIN
  dbms_apply_adm.stop_apply(apply_name =>
'TGT_CDC_APPLY');
END;
/
```

2.13.2 Removing Streams replication completely.

Once the various streams processes have been stopped it is possible to remove them from the systems completely.

2.13.2.1 Drop the propagation

```
BEGIN
  dbms_propagation_adm.drop_propagation(
        propagation_name => 'SRC_TO_TGT',
        drop_unused_rule_sets => true);
END;
/
```

2.13.2.2 Drop the capture

```
BEGIN
  dbms_capture_adm.drop_capture(
        capture_name => 'SRC_CAPTURE',
        drop_unused_rule_sets => true);
END;
/
```

2.13.2.3 Drop the apply

```
BEGIN
  dbms_apply_adm.delete_all_errors(
        apply_name => 'TGT_APPLY');
  dbms_apply_adm.drop_apply(
        apply_name => 'TGT_APPLY',
        drop_unused_rule_sets => true);
END;
/

BEGIN
  dbms_apply_adm.delete_all_errors(
        apply_name => 'TGT_CDC_APPLY');
  dbms_apply_adm.drop_apply(
        apply_name => 'TGT_CDC_APPLY',
        drop_unused_rule_sets => true);
END;
/
```

2.13.2.4 Drop both of the secure queues

Source queue

```
BEGIN
  dbms_streams_adm.remove_queue('SRC_Q1', true,true);
END;
/
```

Destination queue

```
BEGIN
  dbms_streams_adm.remove_queue('TGT_Q1',true,true);
END;
/
```

2.13.2.5 Removing streams configuration

The following procedure removes the streams configuration.

```
BEGIN
  dbms_streams_adm.remove streams configuration;
END;
/
```

2.13.2.6 Drop the streams administrator

The final step is to drop the streams administrator from each system.

```
DROP USER strmadmin cascade;
```

3 MONITORING

There are a number of ways in which the Streams configuration can be monitored. This section describes the alternatives available.

3.1 Strmmon Utility

STRMMON is a monitoring tool focused on Oracle Streams. Using this tool, Database administrators get a quick overview of the Streams activity occurring within a database. The output format comes in two formats: default and long.

The default format reports the rate of activity occurring for Streams processes.

The long format provides the detailed information that was available in previous releases of STRMMON. The reporting interval and number of iterations to display are configurable.

STRMMON can also be used to report Streams activity on two databases at a time within the same strmmon session.

As of Oracle 10g Release 2, STRMMON is distributed as in the demo directory of the database distribution code upon the companion CD.

Note that it tends to display information upon a specific node only, not all the nodes in a RAG cluster.

Strmmon usage

There are 7 command line input parameters for STRMMON: interval, count, user, passw, dbname, sysdba and long. The first 2 parameters (interval and count) control the sampling rate and the amount of output. The next 4 parameters specify the connect information to the particular Streams database. Use multiple occurrences of these 4 parameters to monitor multiple databases within the same strmmon command. Specifying the last parameter (long) displays more detailed information about each process.

When the command **strmmon** is issued without any parameters, a usage message is displayed:

% strmmon

Usage: strmmon -interval <seconds> -count <number> [-user <user name>]

[-passw <password>] [-dbname <database name>] [-sysdba] -long

Parameter Name	Value Units	Description
-interval	Seconds	The interval at which STRMMON will monitor the database. To specify that the sampling rate to be every 3 seconds: -interval 3 This is a required parameter for strmmon.
-count	Number	The number of iterations to monitor the Streams environment. To specify 5 iterations, use the following: -count 5 This is a required parameter for strmmon.
-user	Username	The schema name for logging into the database. Any schema name can be specified. If the SYS schema is specified,

Parameter Name	Value Units	Description
		additional information is displayed. To specify the SYSTEM schema, use -user SYSTEM This parameter should not be specified if logging in as 7 as sysdba" is desired. -user is an optional parameter for strmmon.
-passw	Password	The login password for the schema identified with the -user clause. To specify the password for the SYSTEM schema, use -passw oracle This parameter should not be specified if logging in as 7 as sysdba" is desired -passw is an optional parameter for strmmon.
-dbname	service name	The connection information or service name from tnsnames.ora for the specific database to be monitored. To specify the connect information for the monitored database, use -dbname ORCL.WORLD This is an optional parameter for strmmon.
-sysdba		This flag indicates that the login role is SYSDBA. This optional parameter is typically used with the SYS schema. To specify the login role SYSDBA, use -sysdba When logging in as 7 as sysdba", the –user and -passw parameters are not required.
-long		This flag indicates that the more detailed report is desired. This is an optional parameter for STRMMON. By default, only the capture, apply and propagation rates are displayed.

Table 3 - Strmmon utility parameters

The strmmon output begins with a banner line identifying the program parameters and database. This information is followed with a brief description of the major components of the output display. The Streams Pool Size line is displayed for database versions 10g and above.

Please see the official documentation [Metalink Note: 290605.1] for more details of the usage of this utility.

3.2 DDL Replication

When replicating DDL, the effect of the DDL statement on the replicated sites has to be considered. In particular, one should not allow system generated naming for constraints or indexes, as modifications to these will most likely fail at the replicated site. Also, storage clauses may cause some issues if the target sites are not identical.

When it is decided NOT to replicate DDL in your Streams environment, any table structure change has to be performed manually.

3.3 Propagation

At times, the propagation job may become "broken" or fail to start after an error has been encountered or after a database restart. The typical solution is to disable the propagation and then re-enable it.

```
exec dbms_propagation_adm.stop_propagation('propagation_name');
exec dbms_propagation_adm.startjDropagation('propagation_name');
```

If the above does not fix the problem, perform a stop of propagation with the force parameter and then start propagation again.

```
exec dbms_propagation_adm.stop_propagation('propagation_name',force=>true);
exec dbms_propagation_adm.start_propagation('propagation_name');
```

An additional side-effect of stopping the propagation with the force parameter is that the statistics for the propagation are cleared.

3.4 Source Queue Growth

Source queue may grow if one of the target sites is down for an extended period, or propagation is unable to deliver the messages to a particular target site (subscriber) due to network problems for an extended period. Automatic flow control minimizes the impact of this queue growth. Queued messages (LCRs) for unavailable target sites will spill to disk storage while messages for available sites are processed normally.

Propagation is implemented using the DBMS_JOB subsystem. If a job is unable to execute 16 successive times, the job will be marked as "broken" and become disabled. Be sure to periodically check that the job is running successfully to minimize source queue growth due to this problem.

3.5 Useful tables and views

All Streams processing is done at the "owning instance" of the queue. To determine the owning instance, use the query below:

```
SELECT q.owner, q.name, t.queue_table, t.owner_instance,
FROM DBA_QUEUES q, DBA_QUEUE_TABLES t
WHERE t.object_type = 'SYS.ANYDATA'
AND q.queue_table = t.queue_table
AND q.owner = t.owner;
```

To display the monitoring view information it is possible to either query the monitoring views from the owning instance or use the GV$ views for dynamic streams views.

Dynamic Streams Views	
Streams View Name	Streams View Name from any RAC instance
V$STREAMS_CAPTURE	GV$STREAMS_CAPTURE

Dynamic Streams Views	
V$STREAMS_APPLY_COORDINATOR	GV$STREAMS_APPLY_COORDINATOR
V$STREAMS_APPLY_READER	GV$STREAMS_APPLY_READER
V$STREAMS_APPLY_SERVER	GV$STREAMS_APPLY_SERVER
V$STREAMS_POOL_ADVICE	GV$STREAMS_POOL_ADVICE
V$STREAMS_TRANSACTION	GV$STREAMS_TRANSACTION
V$BUFFERED_PUBLISHERS	GV$BUFFERED_PUBLISHERS
V$BUFFERED_QUEUES	GV$BUFFERED_QUEUES
V$BUFFERED_SUBSCRIBERS	GV$BUFFERED_SUBSCRIBERS
V$PROPAGATION_RECEIVER	GV$PROPAGATION_RECEIVER
V$PROPAGATION_SENDER	GV$PROPAGATION_SENDER
V$RULE	GV$RULE
V$RULE_SET	GV$RULE_SET
V$RULE_SET_AGGREGATE_STATS	GV$RULE_SET_AGGREGATE_STATS
Static Streams Views Capture Views	
DBA CAPTURE	
DBA CAPTURE EXTRA ATTRIBUTES	
DBA CAPTURE PARAMETERS	
DBA CAPTURE PREPARED DATABASE	
DBA CAPTURE PREPARED SCHEMAS	
DBA CAPTURE PREPARED TABLES	
Apply Views	
DBA APPLY	
DBA APPLY CONFLICT COLUMNS	
DBA APPLY DML HANDLERS	
DBA APPLY ENQUEUE	
DBA APPLY ERROR	
DBA APPLY EXECUTE	
DBA APPLY INSTANTIATED GLOBAL	
DBA APPLY INSTANTIATED OBJECTS	
DBA APPLY INSTANTIATED SCHEMAS	
DBA APPLY KEY COLUMNS	
DBA APPLY OBJECT DEPENDENCIES	
DBA APPLY PARAMETERS	
DBA APPLY PROGRESS	
DBA APPLY SPILL TXN	
DBA APPLY TABLE COLUMNS	
DBA APPLY VALUE DEPENDENCIES	
Propagation & Queue Views Streams Views	

Dynamic Streams Views
DBA PROPAGATION
DBA QUEUE SCHEDULES
DBA QUEUE SUBSCRIBERS
DBA QUEUE TABLES DBA QUEUES
Streams Views
DBA REGISTERED ARCHIVED LOG
DBA RECOVERABLE SCRIPT
DBA RECOVERABLE SCRIPT BLOCKS
DBA RECOVERABLE SCRIPT ERRORS
DBA RECOVERABLE SCRIPT PARAMS
DBA STREAMS ADD COLUMN
DBA STREAMS ADMINISTRATOR
DBA STREAMS DELETE COLUMN
DBA STREAMS GLOBAL RULES
DBA STREAMS MESSAGE CONSUMERS
DBA STREAMS MESSAGE RULES
DBA STREAMS NEWLY SUPPORTED
DBA STREAMS RENAME COLUMN
DBA STREAMS RENAME SCHEMA
DBA STREAMS RENAME TABLE
DBA STREAMS RULES
DBA STREAMS SCHEMA RULES
DBA STREAMS TABLE RULES
DBA STREAMS TRANSFORM FUNCTION
DBA STREAMS TRANSFORMATIONS
DBA STREAMS UNSUPPORTED
DBA RULE SET RULES
DBA RULE SETS DBA RULES
DBA HIST BUFFERED QUEUES
DBA HIST BUFFERED SUBSCRIBERS
DBA HIST RULE SET
DBA HIST STREAMS APPLY SUM
DBA HIST STREAMS CAPTURE
DBA HIST STREAMS POOL ADVICE

Table 4 - Dynamic Streams Views

The following tables/views have been found very useful in setting up the streams configuration.

user_rules
all_streams_table_rules
dba_propagation

Dynamic views

gv$streams_capture
gv$streams_apply_reader
gv$streams_apply_server
gv$streams_apply_coordinator

3.6 Monitoring Scripts

To aid in maintenance a number of SQL scripts have been written. These scripts built upon the tables and views described in section 5.5 are described below:

Script Name	Description
apply_conf.sql	Displays Streams Apply configuration
apply_errors.sql	Displays errors encountered in the Streams Apply processes.
apply_progress.sql	Displays the current state of progress of the Streams Apply processes
apply_status.sql	Displays the current Streams Apply status
capture_arch.sql	Displays the details of the archivelogs required to initiate the Capture process.
buffer_q	Displays buffer queue statistics.
capture_config.sql	Displays the Streams Capture configuration.
capture_errors.sql	Displays the Streams Capture errors.
capturejatency.sql	Displays the Streams redo log scanning latency.
capture_perf.sql	Displays the Streams Capture performance.
capture_status.sql	Displays the Streams Capture status.
hc.sql	Oracle supplied Health Check script for Streams 10.2.
pool_advice.sql	Displays the various pool sizes used by Streams configuration.
prop_config.sql	Displays the Streams Propagation configuration.
prop_dest_perf.sql	Displays the Streams Destination Propagation performance.
prop_receiver_status.sql	Displays the Streams Propagation Receiver status.
prop_schedule.sql	Displays the Streams Propagation Schedule.
prop_sender_status.sql	Displays the Streams Propagation sender status.
prop_source_perf.sql	Displays the Streams Propagation Source performance.

Table 5 - Streams Monitorinq SQL scripts

These scripts should be run by a suitably authorised user such as the Streams administrator (strmadmin) or the system user.

3.7 Grid Control Monitoring

Oracle Grid Control provides a convenient method of monitoring (and configuring) streams. The following describes the main screens visible from within Grid Control.

For the example screen shots shown in this document a database performing a capture and propagation mechanism is used.

The first step is to log into the OEM web page using an authorised username and password. Then click on the Targets' Tab (top left hand side of web page) and then the 'Databases' entry item (top right of web page.).

It is expected that the administrator is aware of the source (or destination database) and can pick the appropriate database from the list of all databases available. It is not necessary to pick an individual database instance within a RAC cluster since picking the database name is sufficient.

A database summary page for the chosen database will be displayed as shown in the example below.

Figure 3 - OEM Database home page

To view the streams configuration click on the 'Maintenance' name on the tab bar, and a screen similar to the following will be displayed.

Figure 4 - OEM Database Maintenance page

It is observed that on the left hand side of the page approximately half way down is the Streams heading with two options available 'Setup' and 'Maintenance'. Picking the 'Maintenance option, presents the following page.

Figure 5 - OEM Database Streams Maintenance page.

The screen shows that this chosen database has a single 'Capture' process, a 'Propagation process, no 'Apply' processes and a number of Queue tables and Queues.

Additional information can be obtained upon each of these. Opting for the 'Capture' display presents a screen similar to the following:

Figure 6 - OEM Database Streams Capture page.

Additional information is available for the capture process 'EBA_CAPTURE' by clicking upon the highlighted name presenting the following web page.

Figure 7 - OEM Database Streams Capture Details page.

If instead of opting for the capture process, we had instead opted for the ruleset details we would have been presented with a display similar to the following: [The search for the table to which the rule applied has also been shown in the diagram.]

Figure 8 - OEM Database Streams Capture Rulesets page.

Similarly it is possible to show more details upon the Propagation processes, the apply processes and the messages queue tables and queues. The 'propagation' page display is shown below to compare and illustrate the same page format as explained for the 'capture' process above.

Figure 9 - OEM Database Streams Propagation page.

3.7.1 User Defined Metrics

It is possible within Oracle Grid Control to make use of user defined metrics to aid in monitoring the Streams processes. The following is an example of configuring a user-defined metric to monitor the Streams capture redo log scanning latency using Enterprise Manager:

1. From the Instance Home Page, scroll down to Related Links and click User Defined Metrics.
2. Click Create.
3. To monitor the redo log scanning latency for a particular capture process, specify the following:

   Name: Latency for capture

   STREAMS_CAP1 Type: Number SQL

   Query:

```
SELECT ((SYSDATE - CAPTURE_MESSAGE_CREATE_TIME)*86400)
LATENCY_SECONDS,
FROM V$STREAMS_CAPTORE
WHERE capture_name = 'STREAMS_CAP01';
```

Where the capture name specified is the name of the capture process.

Note: The trick here is to write a query that returns a single value and only one row. It is possible to use any of the queries in the Streams Concepts and Admin guide as long as they are modified to return a single value.

- Database Credentials : Streams administrator user and password
- Thresholds: > 300 (Warning) 900 (Alert)

  (for this query, it is measured in seconds)
- Schedule: set to repeat continuously, at whatever level is best for your business.

Other examples that may be performed include:

- Alerts for spilled messages
- Errors placed in DBA_APPLY_ERRORS
- Status of capture, propagation, or apply is aborted, stopped, or broken The metrics that can be created depend on what it is that needs to be monitored.

3.8 Streams Commander (OSC)

There is another known Streams Monitoring tool known as Oracle Streams Commander. This tool presents a graphical view of the configuration and is reported to be very easy to use. This tool has not been evaluated.

4 AUTOMATING THE SETUP

There is a need to set up all the tables for which the capture process has to run, the tables that need to be propagated (if a schema propagation is not used), the tables for which the replicated apply process is required and the tables for which the change capture process is required.

A simple script similar to that show below can be created for each of the above steps. The example shows the UNIX script for the Streams replicated table apply process.

```
#!/bin/ksh
#
# Script to generate the streams apply script for the Streams replication.
# User is prompted to supply the required details such as database name etc.
# Input list of table names are read from a DNIX file named file_list.
#
# This script is assuming that we are creating a rule for each table,
# and we are changing the schema.
#
# Usage ./gen_apply_scr
# Output is a file named apply.sql
#
# G S Chapman 14th July 2008
#
ofile=apply.sql
# Prompt for required input parameters. Use a default if a carriage return is given.

printf "Please enter source database name (default EBA) "
read sname
if [ -z $sname ] ; then
   # echo "Name not specified"
  sname="EBA"
fi

printf "Please enter destination database name (default  SRC) "
read dname
if [ -z $dname ] ; then
  dname="SRC"
fi

printf "Please enter Streams Administrator (default: STRMADMIN) "
read sadmin
if [ -z $sadmin ] ; then
   sadmin="STRMADMIN"
fi

printf "Please enter source schema (default SB1) "
read sschema
if [ -z $sschema ] ; then
  # echo "Source schema not specified"
  sschema="SBl"
fi

printf "Please enter destination schema (default SB2) "
read dschema
if [ -z $dschema ] ; then
  # echo "Destination schema not specified"
  dschema="SB2"
 fi

printf "Please enter name of input file (default: file_list) "
read flist
if [ -z $flist ] ; then
flist="file_list"
fi

echo "Entered values:"
echo "Source Database name     = " $sname
echo "Destination Database   name = "    $dname
echo "Streams admin            = "        $sadmin
echo "Source schema            = "        $sschema
echo "Destination schema       = "        $dschema
echo "Input file name          = "        $flist
# Check input file specified actually exists.  If not exit,

if [ ! -s $flist ] ; then
```

```
  echo "Specified file name does not exist " $flist
  echo "Job terminating."
  exit 1
fi

# Now let use start generating the script.
printf "DECLARE\n" > $ofile
printf "--\n" >> $ofile  2>/dev/null
printf "   l_dml_rule_name all_rules.rule_name%%TYPE;\n" >> $ofile
printf "    l_dml_rule_name all_rules.rule_name%%TYPE;\n" >> $ofile
printf "--\n" >> $ofile 2>/dev/null
printf "BEGIN\n" >> $ofile
printf "--\n" >> $ofile 2>/dev/null
while read fname do
#
# Check source schema name
# If schema is specified in the input file extract it, otherwise use what was given.
#
schema='echo $fname | awk -F. '{print $1}''
if [ $fname != $schema ] ; then
  ischema=$schema
  if name='echo $fname | awk -F. '{print $2}''
else
 ischema=$sschema
  ifname=$fname fidbms streams adm.add table rules{\n" >> $ofile
$ofile
-\n'
dbms_streams_adm.rename_schema(\n" >> $ofile
rule_name          => l_dml_rule_name,\n" >> $ofile
from_schema_name    => "'$ischema'",\n" >> $ofile
to_schema_name      => "'$dschema'");\n" >> $ofile
\n" » $ofile 2>/dev/null

table_name
streams_type
streams_name
queue_name
include_dml
include_ddl
source_database
inclusion_rule
dml_rule_name
ddl_rule_name
>> S^ofile 2>/dev/null
$ofile
=> "'$ischema"."$ifname"', \n"
=> 'APPLY', \n" >> Sofile
=> ' "'$dname"_APPLY' , \n" >> $ofile
=> "'$sadmin"."$dname"_APPLY_QOEOE' , \n"
=> TRDE,\n" >> $ofile
=> FALSE, \n" >> $ofile
=> "'$sname"', \n">> $ofile
=> TROE,\n" » $ofile
=> l_dml_rule_name, \n" >> $ofile
=> 1 ddl rule name);\n" >> $ofile
printf "END;\n" >> Sofile printf "/\n" >> $ofile
# Now create stop and start scripts
ofile=start_apply.sql
printf "BEGIN\n" > $ofile
printf "—\n" >> $ofile 2>/dev/null
printf "  dbms_apply_adm.start_apply(apply_name =>
printf "—\n" >> $ofile 2>/dev/null
printf "END;\n" >> $ofile
printf "A\n" >> $ofile
"$dname" APPLY');\n" >> $ofile
ofile=stop_apply.sql
printf "BEGIN\n" > $ofile
printf "—\n" » $ofile 2>/dev/null
printf "  dbms_apply_adm.stop_apply(apply_name => "'$dname"_APPLY');\n" » $ofile
printf "—\n" » $ofile 2>/dev/null printf "END;\n" » $ofile printf "A\n" » $ofile
ofile=drop_apply.sql
printf "BEGIN\n" > $ofile
printf "—\n" >> $ofile 2>/dev/null
printf "  dbms_apply_adm.delete_all_errors(apply_name => "'$dname"_APPLY');\n" >> $ofile
printf "  dbms_apply_adm.drop_apply(apply_name => "'$dname"_APPLY',drop_unused_rule_sets
=> true);\n" >> $ofile
printf "—\n" >> $ofile 2>/dev/null
```

```
printf "END;\n" >> $ofile
printf "/\n" >> $ofile
ofile=set_apply_parameters.sql
printf "BEGIN\n" > $ofile
printf "—\n" >> $ofile 2>/dev/null
printf "  dbms_apply_adm.set_parameter(apply_name => '"$dname"_APPLY',
parameter => 'DISABLE_ON_ERROR',
value => 'N');\n" >> $ofile  ~
printf "  dbms_apply_adm.set_parameter(apply_name => '"$dname"_APPLY',
parameter => 'PARALLELISM',
value => '4');\n" >> $ofile
printf "—\n" » $ofile 2>/dev/null
printf "END;\n" >> $ofile
printf "/\n" >> $ofile
```

Similar scripts have been created for the other setup steps indicated. In addition to the prompted inputs there is a file that is required containing the list of schema.tablename pairs.

See Table 7 - Generation Scripts for details of the written scripts and their required input parameters.

Note: The example code above has been formatted to display within this document and may not accurately reflect the format of the actual code used.

5 ADDITIONAL NOTES:

5.1 Database Links

The preferred method of using database links is to initially create a public database link and then create individual private database links using the public database name.

Strictly speaking the most secure is not to have to specify a password for the connection at all, but this is only valid when the connection user/password is available in both source and target database which may not be recommended.

i.e.

```
create public database link TGT using 'TGT';
create database link TGT connect to strmadmin identified by password;
```

The experiences encountered with the streams set up indicated that this preferred mechanism did not seem to work as expected. The links worked as expected for all other encountered situations but not for the STREAMS_ADM setup.

The reason for this is due to the configuration being based upon a RAC cluster with an associated physical standby created by Data Guard. The naming of the each database instance is the name of the database i.e. SRC followed by a letter indicating whether it is the 'live' database 'X' or the physical standby by 'Y'. This gives an instance name of 'SRCX' or 'SRCY'. The fact that it is a RAC cluster results in an additional numberic character being added to each instance name, where 1 is the first instance and "x (x is numeric) to the instance name. i.e. SRCX1, SRCX2 etc.

The set up of the queue on the target node creates a service named 'SYS$STRMADMIN.SRC_Q1.SRC', where 'SRC_Q1 is the queue name and SRC is the database name.

When the propagation mechanism is created it is provided with the name of the destination queue, which is making use of the name of the database link.

i.e.

```
destination_queue_name => 'STRMADMIN.TGT_Q1@TGT'
```

Within the TNSNAMES.ORA entry for the destination database there was specification of the database with the 'live' SERVICE_NAME of 'SRCx'. It is this service name that was used to create the name of the service queue to connect to.

i.e.

```
SYS$STRMADMIN.SRC_Q1.SRCX
```

This connection would never work since the target queue service would not exist.

The solution was to ensure that the TNSNAMES.ORA entry specified a service of SRC, and that such a service was indeed created and running upon the target system. This service is required upon both the 'live' system and the 'standby' system.

For the streams propagation to work successfully it was found that a private database link was required for the streams administrator providing all the required entry, thus not using the public link at all.

5.2 Problems with creating capture process.

A problem may be encountered in creating the capture process. This may be due to running transactions in the system preventing the changing of the database to supplementary logging. This was indeed the situation encountered on one system set up and whilst trying to discover the problem a situation was met where the capture process set up was cancelled and in an 'incomplete state'. This manifested itself in the existence of some of the capture objects but not all.

To resolve the problem the streams administrator was dropped from the database and recreated. This effectively cleared up the database and removed the troublesome objects.

5.3 Database Recovery Scenarios

There may be circumstances where a database may be unavailable due to hardware or other unscheduled outages. Streams implements a comprehensive set of recovery steps to ensure that the integrity of the system is maintained when a database is recovered.

A Streams process will automatically restart after a database restart, assuming that the process was in a running state before the database shut down. No special start-up or shutdown procedures are required in the normal case.

However in some circumstances it may not be possible to perform a full database recovery and a recovery to a point in time is necessary. The steps required in each situation are fully described in the 'Oracle Streams Replication Administrators Guide B14228 Chapter 9'. The interested reader is referred to this document for comprehensive details.

The steps to be followed will depend upon the specific problem encountered by typically these will include the following:

- Point in time recovery on the source in a single source environment
- Point in time recovery in a multi-source environment
- Point in time recovery on a destination database

There are basically two methods that can be used.

The first involves determining the SCN numbers are the source and target database and making changes to which ever database is 'behind' to bring the two into synchronisation. This may involve resetting of the capture process or the creation of an additional process.

The second involves the re-instantiation of the tables, but for a large database this is not a quick method and is not recommended unless the time difference between when the problem occurs and the recovery of the database is large.

Please refer to the official Oracle documentation referenced above for the precise steps to be followed for each scenario.

5.4 Queue Ordering

In 10gR2 onwards commit-time ordering is available. Each message in a commit-time queue is ordered by an approximate commit system change number (approximate CSCN) which is obtained when the transaction that enqueued the message commits.

The COMMIT_TIME choice is a new feature in Oracle Streams AQ 10 g Release 2 (10.2). If it is specified, then any queue that uses the queue table is a commit-time queue, and Oracle Streams AQ computes an approximate CSCN for each enqueued message when its transaction commits.

If COMMIT_TIME is specified as the sort key, then the following must also be specified:

- `multiple_consumers = TRUE`
- `message_grouping = TRANSACTIONAL`
- `compatible =8.1 or higher`

Commit-time ordering is useful when transactions are interdependent or when browsing the messages in a queue must yield consistent results.

5.5 Heartbeat Table

To ensure that the applied_scn of the DBA_CAPTURE view is updated periodically, one recommendation is that a "heart beat" table is implemented. This would only be required on a system that experiences long periods on low activity. A busy system would not require the overhead of an additional table. The streams capture process requests a checkpoint after every 10Mb of generated redo. During the checkpoint, the metadata for streams is maintained if there are active transactions. Implementing a heartbeat table ensures that there are open transactions occurring regularly within the source database enabling additional opportunities for the metadata to be updated frequently. Additionally,

the heartbeat table provides quick feedback to the database administrator as to the health of the streams replication.

5.5.1 To implement a heartbeat table:

Create a table at the source site that includes a date or timestamp column and the global name of the database. Add a rule to capture changes to this table and propagate the changes to each target destination. Make sure that the target destination will apply changes to this table as well. Set up an automated job to update this table at the source site periodically, for example every minute.

5.6 Replicating tables with extra columns

A situation was discovered during development where a table being replicated between 2 systems was found to have an additional column upon the target system. This would obviously requires special handling. The suggested method of handling this type of situation is described below.

The method involved in setting up the replication between the two databases is exactly the same as that described earlier, so for that reason only the additional steps will be documented here.

1. Create streams administrator accounts
2. Create required queues
3. Create capture rules
4. Create propagation rule
5. Instantiate the target objects so that they are the same upon both databases.
6. Create Apply DML handler package
7. Create Apply rules
8. Start up Replication.

Of particular interest is step 6 in the above sequence. For a test environment two tables were created upon the source database, one called SB1_ANY and one called SB1_SPECIAL. Upon the target database the same two tables were created only they are named SB2_ANY and SB2_SPECIAL. In addition the SB2_SPECIAL table has an additional column named 'DELETED_Y' which is a logical delete flag. The intent is that when a row is deleted from the source table that the change is reflected as a logical delete upon the target table by the setting of the additional column to a 'Y' value.

The code generated is illustrated below. Note that there are two procedures within the package. The first is named prc_sb1_rename_any and only performs a rename upon the table object to change the table suffix from 'SB1_' to 'SB2_'. The second procedure named 'PRC_SB1_SPECIAL not only changes the table prefix but also contains the logic to change the delete into an update statement and set the 'DELETED_Y' column value.

```
PACKAGE lcr_handler_pkg
AS
   PROCEDURE prc_sbl_route (pi_lcr_anydata IN SYS.ANYDATA);
   PROCEDURE prc_sbl_rename_any (pi_lcr_anydata IN SYS.ANYDATA);
   FUNCTION fn_sbl_change_name (in_any IN ANYDATA) RETURN ANYDATA;
END lcr_handler_pkg;
/

create or replace PACKAGE BODY lcr_handler_pkg
AS
-- Description: This package is designed to handle the streams apply
-- processes that are special cased.  i.e.  The tables on the source
-- and on the destination are different.
- Input Parameters:
- Output Parameters:
- Error Conditions Raised:
- Author:     G S Chapman
- Revision History
- Date      Author      Reason for Change
```

```
— 29 JUL 2008    G S Chapman  Created.
-- 1 AUG 2008    G S Chapman Add sbl_change_name function.

   PROCEDURE prc_sbl_special (pi_lcr_anydata IN SYS. ANYDATA)
   AS
    -- Procedure for handling SB2_SPECIAL table.
       l_lcr SYS.LCR$_ROW_RECORD;
       l_new_lcr     SYS .LCR$_ROW_LIST;
       l_old_lcr     SYS.LCR$_ROW_LIST;
       l_rc          PLS_INTEGER;
      l_operation    VARCHAR(2) := 'I';
      l_object_owner VARCHAR2(30);
      l_object_name  VARCHAR2(30);
    BEGIN
    -- Get the LCR details and change target schema owner and table name
      l_rc   := pi_lcr_anydata.GetObject (l_l
     l_operation    := UPPER (SUBSTR(l_lcr.get_command_type ( ) , 1, 1) ) ;
      l_object_owner := l_lcr.get_object_owner;
      l_object_name  := l_lcr.get_object_name;
      -- Change objects name to SB2 schema
     l_lcr.set_object_name ( 'SB2_' | | substr (l_object_name, 5) ) ;
      -- Convert DELETE to an UPDATE
      IF l_operation = 'D' THEN
         -- Set old and new values to be the same.
         l_old_lcr := l_lcr .get_values ( 'old' ) ;
         l_new_lcr := l_old_lcr;
        l_lcr.set_values ( 'new' , l_new_lcr) ;
         l_lcr.set_command_type ( 'UPDATE' ) ;

          -- Add new DELETED_Y column values
          -- l_lcr.add_column('old',,'DELETED_Y',NOLL);
         l_lcr.add_column('new','DELETED_Y',anydata.convertvarchar2('Y'));
       END IF;
       l_lcr.execute(true);
   END prc_sbl_special;

    PROCEDURE prc_sbl_rename_any(pi_lcr_anydata IN SYS.ANYDATA)
    AS
    — Procedure for renaming any SBl_table to SB2_table.
       l_lcr SYS.LCR$_ROW_RECORD;
       l_rc  PLS_INTEGER;
       l_object_nameVARCHAR2(30);
    BEGIN
      -- Get the LCR details and change target schema owner and table name
      l_rc   := pi_lcr_anydata.GetObject(l_lcr);
      l_object_name  := l_lcr.get_object_name;
      -- Change object name to SB2 schema
      l_lcr.set_object_name('SB2_'||substr(l_object_name,5)); l_lcr.execute(true);
   END prc_sbl_rename_any;

    FUNCTION fn_sbl_change_name(in_any IN ANYDATA)
       RETURN ANYDATA
    IS
       Icr SYS.LCR$_ROW_RECORD;
       re NUMBER;
       ob_owner VARCHAR2(30);
       ob_name VARCHAR2(30);
    BEGIN
       -- Get the type of object
       -- Check if the object type is SYS.LCR$_ROW_RECORD
       IF in_any.GETTYPENAME='SYS.LCR$_ROW_RECORD' THEN
          -- Put the row LCR into Icr re := in_any.GETOBJECT(Icr);
         -- Get the object owner and name
          ob_owner := Icr.GET_OBJECT_OWNER();
          ob_name  := Icr.GET_OBJECT_NAME();
         IF substr(ob_name, 1, 4) = 'SB1_' THEN
            Icr.set_object_name('SB2_'||substr(ob_name,5));
         END IF;
          Icr.set_object_owner('SB2');
    END IF;
       RETURN ANYDATA.CONVERTOBJECT(Icr);
    END fn_sbl_change_name;
END lcr_handler_pkg;
/
```

The procedure is applied to the apply rules in the usual manner.

Notes:

1) The example code above has been formatted to display within this document and may not accurately reflect the format of the actual code used.

2) There is an alternative manner in which the tables may be renamed and this makes use of a transformation rule. This rule requires a function, shown in the package above 'fn_sbl_change_name'.

5.7 Adding a Global Rule to exclude certain DDL statements

There could also be a need to exclude certain DDL statements from being applied to the Target database like Truncate Table, Grant, Alter trigger etc. This could be achieved by adding a Global Rule on the database where CAPTURE is running. [Be aware that this will turn supplemental logging on Primary Key, Unique Key and Foreign Key at the database level in the SOURCE_DB i.e. implicitly execute "alter database add supplemental log data(PRIMARY KEY.UNIQUE INDEX.FOREIGN KEY) columns" in the Source database].

```
connect STRMADMIN
begin
dbms_streams_adm.add_global_rules(
        streams_type     => 'CAPTURE',
        streams_name     => 'SRC_CAPTURE',
        queuejiame       => 'STRMADMIN.SRC_CAPTURE_QUEUE',
        include_dml      => FALSE,
        include_ddl      => TRUE,
        include_tagged_lcr => FALSE,
        inclusion_rule   => FALSE,
        and_condition    =>'(:ddl.get_command_type()='’CREATE INDEX'’||
                ' or :ddl.get_command_type()='’GRANT'’ ||
                ' or :ddl.get_command_type()='’TRUNCATE TABLE'’||
                ' or :ddl.get_command_type()='’ALTER TRIGGER'’||'));
end;
```

5.8 Excluding Columns in Capture

There may be a need to replicate a table but exclude a specific column. This can be achieved by the use of a specific rule. In the following example a BLOB column is excluded from the capture process.

```
BEGIN
   DBMS_STREAMS_ADM.DELETE_COLUMN(
      rule_name          =>
'STRMADMIN.DELETE_COLUMNS',
      table_name         =>
'SCOTT.WITH_BLOB',
      column_name   => 'BLOB_DATA',
      value_type         => '*',
      step_number   => 0,
      operation          => 'ADD');
END;
/
```

Where the rule name is assumed to be 'DELETE_COLUMNS'. To determine the specific rule name in use it is necessary to perform a select upon the DBA_RULE table. Remember that if the capture database is also propagation that there will be double the number of

rules that might be expected. It is only necessary to apply the rule to the 'CAPTURE' process.

A1 STREAMS UTILITY PACKAGE

A utility package has been written and is usually located under the Streams Administration user. This package named STRMUTIL_PKG consists of a number of procedures to avoid having to have a number of individual scripts around. The procedures are described in the table below.

Procedure Name	Input Parameters	Parameter Description	Comments
prc_stop_capture	p_capture_name	IN VARCHAR2 DEFAULT NULL	If the input parameter is specified stop the specified capture process if it is enabled. If no input parameter is specified stops all capture processes running upon the database that are enabled.
prc_stop_apply	p_apply_name	IN VARCHAR2 DEFAULT NULL	If the input parameter is specified stop the specified apply process if it is enabled. If no input parameter is specified stops all apply processes running upon the database that are enabled.
prc_start_capture	p_capture_name	IN VARCHAR2 DEFAULT NULL	If the input parameter is specified stop the specified capture process if it is disabled. If no input parameter is specified starts all capture processes upon the database that are disabled.
prc_start_apply	p_apply_name	IN VARCHAR2 DEFAULT NULL	If the input parameter is specified stop the specified apply process if it is disabled. If no input parameter is specified starts all apply processes upon the database that are disabled.
prc_instantiate	p_dbname	IN VARCHAR2 DEFAULT 'EBA'	Instantiates all objects defined in the streams rules upon the connected database. NOTE: If no input parameter is supplied a default database name of EBA is assumed.
prc_add_ct_cols	p_owner	IN VARCHAR2	When provided with a valid owner and table name will add the additional columns required for the Change Capture apply process.
	p_table_name	IN VARCHAR2	
prc_stop_propagation	p_prop_name	IN VARCHAR2 DEFAULT NULL	If the input parameter is

Procedure Name	Input Parameters	Parameter Description	Comments
	p_owner	IN VARCHAR2 DEFAULT 'STRMADMIN'	specified stop the specified propagation if it is enabled. If no input parameter is specified stops all propagation processes running upon the database that are enabled.
prc_start_propagation	p_prop_name	IN VARCHAR2 DEFAULT NULL	If the input parameter is specified stop the specified propagation process if it is disabled. If no input parameter is specified starts all propagation processes upon the database that are disabled.
	p_owner	IN VARCHAR2 DEFAULT 'STRMADMIN'	

Table 6 - STRMUTIL_PKG procedures.

This package may evolve over time so please check the version installed upon the database that is being used.

A2 GENERATION SCRIPTS

A number of UNIX scripts have been written to automate the generation of SQL scripts to create the Streams entries. An example of one of the scripts is shown in section 4.

Script Name	Inputs	Defaults (see Note 3)	Comments
gen_apply_schema_scr	Source database name	SRC	Creates a script named 'apply.sql' for 'basic' schema replication apply.
	Destination database name	TGT	
	Streams administration user	STRM ADMIN	
	Source schema name	SB1	
	Destination schema name	SB2	
gen_apply_tables_scr	Source database name	SRC	Creates a script named 'apply.sql' to create the 'basic' application for the replicated tables
	Destination database name	TGT	
	Streams administration user	STRMADMIN	
	Source schema name	SB1	
	Destination schema name	SB2	
	Input file name	Filelist	
gen_capture_tables_scr	Source database name	SRC	Creates a script named 'capture.sql' for the capture of tables. Note: Source schema name is provided only in case the schema is not specified in the input file. Target database name is used to create instantiate script.
	Streams administration user	STRMADMIN	
	Source schema name	SB1	
	Target database name	TGT	
	Input file name	Filelist	
gen_capture_schema_scr	Source database name	SRC	Creates a script named 'capture.sql' for the capture of all tables of the specified schema. Target database name is used to create instantiate script.
	Streams administration user	STRMADMIN	
	Source schema name	SB1	
	Target database name	TGT	
gen_cdc_apply_tables_scr	Source database name	SRC	Creates a script named 'cdc_apply.sql' for the CDC table rules. Assuming no schema renaming is required.
	Destination database name	TGT	
	Streams administration user	STRMADMIN	
	User procedure	CDC HANDLER	
	Input file name	Filelist	
gen_cdc_apply_schema_scr	Source database name	SRC	Creates a script named 'cdc_apply.sql' for the CDC table rules. Assuming schema renaming is required.
	Destination database name	TGT	
	Streams administration user	STRMADMIN	
	Source schema	SB1	

Script Name	Inputs	Defaults (see Note 3)	Comments
	Destination schema	SB2	
	User procedure	CDC HANDLER	
	Input file name	Filelist	
gen_propagation_scr	Source database name	SRC	Generates script named 'propagation. sql' to create propagation rules
	Destination database name	TGT	
	Source schema	SB1	
	Streams administration user	STRMADMIN	
gen_queue_scr	Source database name	SRC	Generates two scripts named 'src_queue.sql' and dest_queue.sql' to create queues on both source and destination database.
	Destination database name	TGT	
	Streams administration user	STRMADMIN	
run_impdp	None	N/A	Script to read the parameter files (PARFILE1, PARFILE2 or PARFILE) to load the target database objects

Table 7 - Generation Scripts

Notes:

- For those scripts that accept a file as a parameter, the file should be formatted as a simple text file, with one entry per line, left justified, and each entry will comprise a schema name, followed by a full stop and then the table name.

  i.e. AAA.TABLE_ONE

  BBB.TABLE_TWO

- If the schema name and full stop are not specified the table name will be assumed to be as specified by the source schema, where this has been provided as an option.
- Where more that one schema is to be propagated, some of the scripts need to be run more than once. An example would be the gen_propagation script. After running once for a specific schema, one should run the generated output script into the database. Then rerun the generation script again for the next schema, repeating the running of the generated output script into the database. See the readme.txt file in each directory for more details.
- The defaults provided above will vary depending upon which configuration is being set up. The default values reflect the expected databases which will be involved in the Streams replication.
- The run_impdp script reads supplied parameter files. These parameter files need editing to ensure that the correct mapping of schemas is performed when loading the tables into the target database from the source database when first setting up the configuration.

A3 USAGE OF WRITTEN SCRIPTS.

To set up the Streams Replication environment the following scripts need to be run:

1. Configure database initialisation parameters on both source and target databases
2. Create streams administrator on both Source and Target databases.
3. Run 'gen_queue_scr' script described above to create two scripts that are then run. One upon the source database and the second on the target database.
4. As the streams administrator, create a database link between the source and the target database from the source database.
5. As the streams administrator run the strmutil_pkg script to create the Streams utility package. Permissions to use the package need to be granted when it is determined who needs access.
6. Run the 'gen_propagation_scr' script to create the 'propagation.sql' which is then run on the source database. If more than one schema is involved run the script multiple times providing the details of each required schema to be propagated.
7. Run the alter_propagation script to change the propagation processing parameters.
8. Run the 'gen_capture_tables_scr' script or the gen_capture_schema_scr script to create the capture.sql script which is then run on the source database. The script to run will depend upon whether an entire schema is being captured or specific tables. It is possible of course that both, might apply, in which case run one, and then execute the capture.sql script in the database, before running the second.
9. Run the alter_capture script to change the processing parameters.
10. Run the 'gen_apply_scr' OR the 'gen_apply_schema_scr' script to create the apply.sql script which is then run upon the target database. The choice of script will depend upon the number of tables involved. If multiple schemas are involved run for each schema supplying the required parameters each time.
11. Run the set_apply_parameters script to change apply processing parameters.
12. If Change tables are not being created from the LCR records go to step 16 below.
13. Create the CDC insert package upon the target database.
14. Run the 'gen_cdc_apply_scr' script to create the cdc_apply.sql script which can then be run upon the target database.
15. Run the set_cdc_apply_parameters script to change the cdc apply processing parameters.
16. Load the target database with all base tables which are being replicated. The impdp command is most useful for small schemas.
17. Run the 'instantiate.sql' script which was created in step 8 above on the source database.
18. If schema capture is being used then run the exclusion.sql script on the source database. This was created in step 8 above.
19. Run the start apply processes for both the normal apply and the cdc_apply on the target database.
20. Run the start capture process upon the source database.
21. Run the start propagation process upon the source database.

A4 UPDATE RELEASE CHANGES

A new release of the application will often involve the creation of modification of a number of tables. New tables may be created, old ones removed and other may be modified. Each of these changes will require a number of steps to be performed. These can be described broadly as below:

1. Bring the databases to a quiet state with no other user activity being performed.
2. Ensure all capture and apply operations have been completed.
3. Stop the capture process
4. Stop the propagation process
5. Stop the apply process(es).
6. Create any new tables on both the source and target databases.
7. Modify any tables which require the adding or removal of columns or definitions as appropriate.
8. If schema propagation in being used there is nothing more to do with propagation. If table propagation is being used, add the new tables to the propagation rules, and remove and deleted tables from the propagation rules.
9. Change the capture rule to add any new tables required to be replicated and remove any tables which will no longer exist.
10. If a table apply run is being used then add the new tables to the rule, and remove any un-required or deleted tables from the rule.
11. If a Change capture rule is being used, modify the rule to add or remove tables as appropriate.
12. In the event of a change to the Change capture table structure, modify the change table and also the change package procedure as appropriate.
13. Instantiate all new tables and modified tables so that the source and target databases are synchronised. One may want to instantiate all objects.
14. If a new schema is being introduced, modify propagation and apply rules to add the details of the new schema.
15. Start up the propagation process
16. Start up the apply process
17. Start up the capture process

A5 ADDING A FEW TABLES TO THE REPLICATION.

There is a need sometimes to add extra tables to an existing streams replication setup. This is a reasonably common occurrence in the development environment. This readme describes the steps to be performed.

1. Identify the tables to be added.
2. Stop the apply process on the target database.
3. Stop the capture process on the source database.
4. Ensure that there is an up to date copy of the table(s) on the target database. A simple CTAS [Create table as Select] statement is usually sufficient for a single or very few tables.
5. Create an instantiation script to run on the source database for the table(s).
6. IF this is a table capture, create a small script to add the new table(s) to the capture process.
7. IF this is a schema capture there is no need to do anything special.
8. Restart the capture process
9. Restart the apply process.
10. Check that the propagation process is running.
11. Add the select table grants to the schema role on the target database.
12. Create a synonym for the accessor schema for the new table(s).
13. Check everything is working OK
14. All done.

Issues and Advanced Topics

A6 HANDLING APPLY SPILL

A problem has been encountered that resulted in the apply process being blocked. This resulted in a large amount of disk space being occupied by the Apply Spill files. The following is obtained from Oracle and describes how the Apply Spill areas are purged in Oracle Release 10.2.

A6.1 Apply Spill (10.2)

Apply Spillover, introduced in 10.2, allows Streams to handle large transactions and long running transactions. A large transaction is a transaction with more than 10000 Logical Change Records (LCRs). The size of a large transaction can be set by the user via the apply parameter TXN_LCR_SPILL_THRESHOLD. LCRs will start to spill to the apply spill table when the number of LCRs in the transaction reaches the value of TXN_LCR_SPILL_THRESHOLD. The apply spill table is located by default in SYSAUX and uses a partitioned table to store the LCRs. Each transaction is a separate partition of the table (the key is apply_name and transaction id). When a transaction has been applied and it is time to remove the LCRs for that transaction, a drop partition is performed for the transaction id. This is a very quick operation and the space in SYSAUX is returned. If an error should occur while applying the transaction, the transaction will remain in the apply spill table and be marked as an error.

A6.2 Purging Apply Spill

Note this procedure should only be used under the guidance of Support. Purging a transaction from the apply spill table can result in data inconsistencies in the user tables (ORA-1403 or other apply errors). This should not be done unless the customer completely understands the impact of this action. When the fix for unpublished Bug 5736709 is implemented _ignore_transaction will be sufficient to ignore any kind of transaction whether it is spilled or not.

To purge a transaction from the apply spill table, use the following process

1. Obtain the PLB file: streams_purge_apply_spill_txn.plb from Oracle. Note that this is not available except upon specific request.
2. Load this procedure into the SYS schema for the target database using sqlplus. This creates the procedure purge_spill_txn in the SYS schema.

   ```
   connect / as sysdba @streams_purge_apply_spill_txn.plb
   ```

   The procedure signature is

   purge_spill_txn(apply_name , ignore_txn_id , delete_only)

   where the parameters have the following meaning:

   apply_name> is the name of the apply process for the spilled transaction to be ignored.

   ignore_txn_id :- is the transaction id of the spilled transaction.This transaction id must also be listed in the _ignore_transaction parameter of the apply process.

   delete_only :- is a boolean value. The default is FALSE which will drop the partition of the spilled transaction. Set to TRUE if you just want to remove the spilled transaction.
3. Check the DBA_APPLY_SPILL_TXN view to identify the transaction id to remove

   ```
   select apply_name, xidusn]|'.'||xidslt||'.'||xidsqn
   txn_id, first_scn, first_message_create_time,
   message_count, spill_creation_time
   from dba_apply_SPILL_TXN;
   ```

4. Stop all apply processes. For this example, we have one apply process and the apply name is APP_TEST.

```
exec dbms_apply_adm.stop_apply('APP_TEST');
```

5. Set the apply parameter _IGNORE_TRANSACTION with the relevant transaction id from step 3.

For this example, the transaction id is 10.22.333.

```
exec dbms_apply_adm.set_parameter('APP_TEST',
'_ignore_transaction','10.22.333');
```

6. Run the purge_spill_txn procedure supplying the apply name and the transaction id.

```
exec sys.purge_spill_txn('APP_TEST', '10.22.333');
```

7. Check the DBA_APPLY_SPILL_TXN view to confirm that the transaction has been removed.

```
SELECT * FROM DBA_APPLY_SPILL_TXN
where transaction_id = '10.22.333';
```

8. Clear the transaction id from the apply parameters

```
exec dbms_apply_adm.set_parameter('APP_TEST',
'_ignore_transaction',null);
```

9. Restart all the apply processes

```
exec dbms_apply_adm.start_apply('APP_TEST');
```

A7 PROBLEMS RESTARTING CAPTURE PROCESS

A problem was encountered restarting a Capture process. The symptoms were that the capture process appeared stuck in the step of mining the redo logs.

The information given here is extracted from Metalink Note 471695.1 and can also be used to move forward past a missing logfile.

A new Capture process can start from an existing Streams Dictionary build or from a new build. If no explicit build is performed, creating a Capture process will perform this operation implicitly.

In order to determine the Steams Dictionary builds that exist issue:

```
column first_change# heading 'First SCN' format 9999999999999
column next_change# heading 'First SCN' format 9999999999999
column name heading 'Log File Name' format A50

select distinct first_change#,next_change#, name
from v$archived_log
where dictionary_begin = 'YES'
order by first_change#;
```

If a Dictionary build is available, the first_change# of the related log can be used as the first_scn value (step 2 is therefore not necessary) as detailed in step 4 below.

The Steps to recreate the Capture process are as follows:

1. Drop the current capture process:

   First of all record relevant information which should be considered in step 4 below.

```
select queue_name, capture_name, rule_set_name, rule_set_owner,
                source_database,negative_rule_set_name,
               negative_rule_set_owner, checkpoint_retention_time
from dba_capture queue_name = '<queue name>'
and queue_owner = '<owner>';
```

(Note: column checkpoint_retention_time is not present in 10.1).

then drop the capture process :

```
exec dbms_capture_adm.stop_capture('<capture name>');
exec dbms_capture_adm.drop_capture('<capture name>');
```

2. Generate a new dictionary dump in the current log :

```
set serveroutput on
declare
  sen number;
begin
  dbms_capture_adm.build(first_scn => sen);
  dbms_output.put_line('First SCN Value = ' || sen);
end;
/
```

Note: please record the first SCN Value.

3. Ideally, database objects should be prepared for instantiation after a build is performed.

   Run one or more of the following procedures in the dbms_capture_adm package to prepare database objects for instantiation:

prepare_global_instantiation

prepare_schema_instantiation

prepare_table_instantiation

1. Create the capture

   process, e.g. :

2.

```
begin
dbms_capture_adm.create_capture(
queue_name => '<queue owner>.<queue name>',
capture_name => '<capture process name>',
rule_set_name => '<exiting ruleset used by capture process>',
first_scn => &no, — <enter the value for the first scn>);
end;
/
```

The above is an example. You should also consider whether details from Step 1 are also relevant: negative ruleset, etc.

5. If required, re-instantiate the replicated tables at the destination, either manually using exp/imp or expdp/impdp and then setting the correct instantiation sen using:

   dbms_streams_adm.set_<table|schema>_instantiation_scn().

   Ensure that the flashback_scn on export / import is the same as that used on the relevant dbms_streams_adm.set_<table|schem> instantiation_scn() call. This will ensure that the destination database will only apply changes after the instantiation sen. The sequence is therefore as follows :

   ```
   # on the source database
   column inst_scn format 9999999999999
   select dbms_flashback.get_system_change_number() inst_scn from dual;

   # move the data across ; use the inst_scn as the flashback_scn on export export / import data

   # against the target database; use the inst_scn as the instantiation_scn: use
   dbms_apply_adm.set_schema_instantiation_scn /
   dbms_apply_adm.set_table_instantiation_scn
   ```

6. Restart the Apply process first then start the new Capture process.

A8 KNOWN PROBLEMS WHEN REMOVING STREAMS CONFIGURATION

10.1 introduced the procedure dbms_streams_adm.remove_streams_configuration to simplify the Streams cleanup. This procedure removes the Streams configuration at the local database. In addition, doing a DROP USER... CASCADE will work, as long as you have stopped the Streams processes. However there are still some elements of the configuration that remain even after running these steps. These are listed below:

1) DML Handlers are not removed

After running dbms_streams_adm.remove_streams_configuration and dropping the streams administrator (STRMADMIN), the apply handlers are still seen in dba_apply_dml_handlers.

These are known bugs which as far as I can tell still seem to be present in 10.2.0.4:

Bug 4772753 REMOVE_STREAMS_CONFIGURATION DOES NOT REMOVE DML HANDLERS

Bug 2284725 NEED TO REMOVE ROWS FROM DICTIONARY VIEW WHEN DML HANDLE IS NULLED OUT

2) Buffered Queue and Agents not cleared

The remove_streams_configuration package leaves entries for: Streams Buffered Queue History for last day Streams Buffered Subscriber History for last day Agents Agent Privileges

The Agents can be removed by using the DBMS_AQADM.DISABLE_DB_ACCESS or DBMS_AQADM.DROP_AQ_AGENTS procedures. The history information is stored as part of the workload repository snapshots. The retention is controlled by using the DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS.

3) Entries for dbms_apply_adm.compare_old_values dba_apply_table_columns still show values.

The DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER procedure can be used to set the method_name to NULL with the originally specified object_name, columnjist, and resolution_column removed methods. If NULL, then the procedure removes any existing update conflict handler with the same object_name, resolution_column, and columnjist. If non-NULL, then the procedure replaces any existing update conflict handler with the same object_name and resolution_column.

A9 USING STREAM TAGS

There may be a need to perform some DML (or DDL) activities upon a source database for which there is a desire to prevent the action from being 'applied' upon the target database. Streams provides the ability to do this but the use of 'stream tags' which get applied to the LCR record.

By default streams are set up to ignore any tags upon LCR records. The main use of tags is in a multiple database environment where changes get replicated to two or more other database. In these situations it is important that changes only get applied once at each database.

It is possible to set or get the value of the tags generated by the current session or by an apply process. The main interest here is for the setting of a tag at the current session.

A9.1 Setting the Tag Values Generated by the Current Session

You can set the tag for all redo entries generated by the current session using the SET_TAG procedure in the DBMS_STREAMS package. For example, to set the tag to the hexadecimal value of ' ID ' in the current session, run the following procedure:

```
BEGIN
  DBMS_STREAMS.SET_TAG(tag  =>  HEXTORAW('ID')};
END;
/
```

After running this procedure, each redo entry generated by DML or DDL statements in the current session will have a tag value of ID. Running this procedure affects only the current session.

The following are considerations for the SET_TAG procedure:

- This procedure is not transactional. That is, the effects of SET_TAG cannot be rolled back.
- If the SET_TAG procedure is run to set a non-NULL session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the SET_TAG procedure in a session. A data dictionary build happens when the DBMS_CAPTURE_ADM. BUILD procedure is run. The BUILD procedure can be run automatically when a capture process is created.

A9.2 Getting the Tag Value for the Current Session

You can get the tag for all redo entries generated by the current session using the GET_TAG procedure in the DBMS_STREAMS package. For example, to get the hexadecimal value of the tags generated in the redo entries for the current session, run the following procedure:

```
SET SERVEROUTPUT ON
DECLARE
  raw_tag RAW(2048);
BEGIN
  raw_tag := DBMS_STREAMS.GET_TAG();
  DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

You can also display the tag value for the current session by querying the DUAL view:

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

A9.3 Example using Tags

The following example illustrates a real situation where the insert of the employee with a number of 1000 is not replicated from the source to the target database.

```
begin
  dbms_streams.set_tag(tag => hextoraw('ID'));
  insert into scott.emp
  values (1000,'ZZZZ',lMGRl,7521,sysdate,1000,NULL,10);
  dbms_streams.set_tag(tag => null);
  insert into scott.emp
  values (1001,'YYYY','MGR',7251.sysdate, 10OO.NULL, 10);
  commit;
end;
/
```

A10 EXAMPLE OF DROPPING COLUMNS AND STREAMS IN A SINGLE DATABASE

The following example illustrates two specific aspects of streams replication. The first is the replication with a single database. It shows a simple table copy from one schema to another. The second aspect is the dropping of a column in a table that is being replicated. The removal of the column is performed on the capture side so that the column data is not replicated. The chosen column is in this example a BLOB, but could be any of the permitted data types.

```
-- Create users
create user geoffc1 identified by abcd;
grant resource to geoffc1;
create user geoffc2 identified by abcd;
grant resource to geoffc2;

-- Create test tables
CREATE TABLE geoffc2.with_blob (a NUMBER PRIMARY KEY,
     c VARCHAR2(20));

CREATE TABLE geoffc1.with_blob (a NUMBER PRIMARY KEY,
     b BLOB,
     c VARCHAR2(20));

CREATE TABLE geoffc2.without_blob (a NUMBER PRIMARY KEY,
          c VARCHAR2(20));

CREATE TABLE geoffc1.without_blob (a NUMBER PRIMARY KEY,
          c VARCHAR2(20));

-- Create queues
BEGIN
  dbms_streams_adm.set_up_queue(
     queue_table     => 'GEOFF_QT_2A',
     queue_name     => 'GEOFF_CAPTURE_QUEUE',
     queue_user                => 'STRMADMIN');

     dbms_streams_adm.set_up_queue(
     queue_table               => 'GEOFF_QT_2B',
     queue_name     => 'GEOFF_APPLY_QUEUE',
     queue_user                => 'STRMADMIN');
END;
/

-- Check the queues are set up
select * from dba_queues;

-- Set up propagation process rule
BEGIN
  dbms_streams_adm.add_schema_propagation_rules(
     schema_name               => 'GEOFFC1',
     streams_name              => 'GEOFF_INTERNAL',
     source_queue_name     => 'STRMADMIN.GEOFF_CAPTURE_QUEUE',
     destination_queue_name          => 'STRMADMIN.GEOFF_APPLY_QUEUE',
     include_dml                     => TRUE,
     include_ddl                     => FALSE,
     source_database                 => 'EBR',
     inclusion_rule            => TRUE,
     queue_to_queue                  => TRUE);
END;
```

```
/
-- Set up the apply process
DECLARE
  l_dml_rule_name all_rules.rule_name%TYPE;
  l_ddl_rule_name all_rules.rule_name%TYPE;
BEGIN
  dbms_streams_adm.add_schema_rules(
    schema_name  => 'GEOFFC1',
    streams_type   => 'APPLY',
    streams_name  => 'GEOFF_APPLY',
    queue_name    => 'STRMADMIN.GEOFF_APPLY_QUEUE',
    include_dml              => TRUE,
    include_ddl              => FALSE,
    source_database          => 'EBR',
    inclusion_rule   => TRUE,
    dml_rule_name => l_dml_rule_name,
    ddl_rule_name  => l_ddl_rule_name);
  dbms_streams_adm.rename_schema(
    rule_name                       => l_dml_rule_name,
    from_schema_name                => 'GEOFFC1',
    to_schema_name                  => 'GEOFFC2');
END;
/
-- Create the capture rules.
BEGIN
  dbms_streams_adm.add_table_rules(
    table_name               => 'GEOFFC1.WITH_BLOB',
    streams_type   => 'CAPTURE',
    streams_name  => 'GEOFF_CAPTURE',
    queue_name    => 'STRMADMIN.GEOFF_CAPTURE_QUEUE',
    include_dml              => TRUE,
    include_ddl              => FALSE,
    inclusion_rule   => TRUE);

  dbms_streams_adm.add_table_rules(
    table_name               => 'GEOFFC1.WITHOUT_BLOB',
    streams_type   => 'CAPTURE',
    streams_name  => 'GEOFF_CAPTURE',
    queue_name    => 'STRMADMIN.GEOFF_CAPTURE_QUEUE',
    include_dml              => TRUE,
    include_ddl              => FALSE,
    inclusion_rule   => TRUE);
END;
/

-- Instantiate the objects.
DECLARE
vlscn NUMBER;
BEGIN
 vlscn := dbms_flashback.get_system_change_number();
 dbms_apply_adm.set_table_instantiation_scn(
   source_object_name              => 'GEOFFC1.WITH_BLOB',
   source_database_name  => 'EBR',
   instantiation_scn          => vlscn);
END;
/

DECLARE
vlscn NUMBER;
BEGIN
```

```
  vlscn := dbms_flashback.get_system_change_number();
  dbms_apply_adm.set_table_instantiation_scn(
    source_object_name                => 'GEOFFC1.WITHOUT_BLOB',
    source_database_name  => 'EBR',
    instantiation_scn             => vlscn);
END;
/
-- Determine which rule has to have the delete column applied to.
select rule_name, rule_condition from dba_rules;

-- Add the rule to delete the column to the capture process.

-- We will assume that the rule has been determined as WITH_BLOB82 from the select
script
-- just run.
-- If not modify the following code to ensure that the correct rule is specified.
BEGIN
  DBMS_STREAMS_ADM.DELETE_COLUMN(
     rule_name                  => 'STRMADMIN.WITH_BLOB82',
     table_name                 => 'GEOFFC1.WITH_BLOB',
     column_name   => 'B',
     value_type                 => '*',
     step_number    => 0,
     operation                  => 'ADD');
END;
/

-- We can now start up the apply and capture processes.
BEGIN
  dbms_apply_adm.start_apply(apply_name => 'GEOFF_APPLY');
END;
/

BEGIN
  dbms_capture_adm.start_capture(capture_name => 'GEOFF_CAPTURE');
END;
/

-- Check the capture view until the state changes to ‘CAPTURING CHANGES’.
-- This may take a short while.
select * from gv$streams_capture;


-- Now we can start putting data into the tables.

select * from gv$streams_capture;

-- Now insert some real data into the source tables.

insert into geoffc1.without_blob
values (3, 'Test 3');


insert into geoffc1.with_blob
values (1, empty_blob(), 'Test 1');

insert into geoffc1.with_blob
values (2, empty_blob(), 'Test 2');

-- Check the results
```

```
select * from geoffc2.with_blob;
select * from geoffc1.with_blob;

select * from geoffc2.without_blob;
select * from geoffc1.without_blob;

-- The inserted data should now be visible in the GEOFF2 schema tables.

-- Clean up after our test.
BEGIN
  dbms_apply_adm.stop_apply(apply_name => 'GEOFF_APPLY');
END;
/

BEGIN
  dbms_capture_adm.stop_capture(capture_name => 'GEOFF_CAPTURE');
END;
/

BEGIN
  dbms_propagation_adm.stop_propagation(propagation_name =>
'GEOFF_INTERNAL');
END;
/
BEGIN
  dbms_capture_adm.drop_capture(capture_name =>
'GEOFF_CAPTURE',drop_unused_rule_sets => true);
END;
/

BEGIN
  dbms_apply_adm.delete_all_errors(apply_name => 'GEOFF_APPLY');
  dbms_apply_adm.drop_apply(apply_name =>
'GEOFF_APPLY',drop_unused_rule_sets => true);
END;
/

BEGIN
  dbms_propagation_adm.drop_propagation(
    propagation_name              => 'GEOFF_INTERNAL',
    drop_unused_rule_sets => true);
END;
/

BEGIN
  dbms_streams_adm.remove_queue(
    queue_name            => 'GEOFF_CAPTURE_QUEUE',
    cascade                       => TRUE,
    drop_unused_queue_table       => TRUE);

  dbms_streams_adm.remove_queue(
    queue_name            => 'GEOFF_APPLY_QUEUE',
    cascade                       => TRUE,
    drop_unused_queue_table       => TRUE);
END;
/
```

A11 TRACING STREAMS

To trace the propagation process use the following event code:

Event 24040 is used for tracing the propagator.

Level 1 - Trace unexpected **event**s (e.g. rejecting expired msgs, errors)
Level 2 - 1 plus Trace entry and exit from C callout
Level 3 - 2 plus trace result from each loop to send messages
Level 8 - 7 plus trace TOIDs of target qs when fetched from dictionary
Level 9 - 8 plus trace all of **propagation**' target queues
Level 10 - 9 plus trace message ids dequeued locally and enqueued remotely

High Availability Deployment

A12 DEPLOYING STREAMS IN A HIGH AVAILABILITY ARCHITECTURE.

The following considerations must be taken into account while deploying stream in high availability architecture.

- As standby by database is an exact duplicate of primary database, so it must be setup with all aspects of a Streams configuration such as **capture process, propagation rules, and apply services.**

- In order to provide seamless replication through the switchover, the source primary and standby hosts must have identical Oracle Net configuration files **(tnsnames.ora).** This assures that all **dblinks** used by the Streams configuration can correctly resolve Oracle Net service names.

- A Streams configuration requires supplemental logging. The same supplemental logging that exists on the primary must be enabled on the physical standby.

  a) To verify the current state of supplemental logging on the standby run the following query:

```
SELECT supplemental_log_data_min,
       supplemental_log_data_pk,
       supplemental_log_data_ui,
       supplemental_log_data_fk,
       supplemental_log_data_all
FROM   v$database;
```

  b) If the supplemental logging was enabled at the Primary after the Physical standby was created then enable supplemental logging at the database level as follows:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG
DATA (PRIMARY KEY.UNIQUE INDEX.FOREIGN KEY.ALL) COLUMNS;
```

  c) Table level supplemental logging does generate redo and will be sent to the physical standby. To verify table logging run the following query on primary and standby instance.

```
SELECT owner,
       table_name,
       log_group_name,
       log_group_type,
       always,
       generated
FROM   all_log_groups;
```

Note: Refer to the 'Streams Setup' guide for more information.

A12.1 Procedure to recover from a Streams destination failure

This procedure provides guide lines as to how to deal with situation of any accidental outage or catastrophic failure of a Streams destination database. To deal with this situation if a business decision requires a role transition, then we need to make sure that all streams source and

destination are moved over to the DR site using the appropriate switchover and failover procedures described earlier.

This situation needs the following action:

a) Perform failover operation on Streams destination database.

b) Perform switchover operation on Streams source database.

A12.2 Perform the following steps to recover a Streams destinations failure.

Streams Destination Failover to DR site		
1	Performs Streams destination failover. Note: After a successful failover, the streams destination standby database transitions to the primary role and old primary becomes new standby.	Follow the generic failover procedure described earlier to switch Streams Destination database.
2	Verify the status of propagation on the Streams source database.	Login as streams administrator on the source database and check the status as follows: connect STRMADMIN/<Password> SELECT propagation_name,status FROM dba_propagation;
3	Stop propagation on the source database if already enabled. Note: If any problem encountered the propagation should be dropped.	Login as streams administrator on the source database and stop propagation. connect STRMADMIN/<Password> execdbms_propagation_adm.stop_propagation (propagation_name => '<propagation Name>'); Note: If there are problem stopping propagation, then it can be dropped as follows: EXECdbms_propagation_adm.drop_propagation (propagation_name => '<Propagation Name>', drop_unused_rule_sets => TRUE);
4	Perform a query on the new destination database to determine the highest applied SCN (oldest message number) and make a note of it. Note: HIGHEST APPLIED SCN: _____	Login as stream administrator on the new destination and determine the highest applied SCN. SELECT oldest_message_number FROM dba_apply_progress;

Streams Destination Failover to DR site		
5	Stop the capture process on the source database if enabled.	Login as stream administrator on the source database and stop the capture process as follows: connect STRMADMIN/<Password> • Verify the status of the capture process as follows: SELECT capture_name,status FROM dba_capture; • If capture process is already enabled then stop the capture process as follows: EXEC dbms_capture_adm.stop_capture (capture_name => '<Capture Name1);
6	Reset the start SCN for the capture process on source database to the **HIGHEST APPLIED SCN** value noted in step-4.	Login as stream administrator on the source database and reset the start SCN of capture process as follows: EXEC dbms_capture_adm.alter_capture (capture name => '<Capture Name>', start_scn => < HIGHEST APPLIED SCN in step 4>);
Streams Source switchover to DR site		
7	Perform a switch over of the Streams source primary database. Note: After a successful switchover, the streams source standby database transitions to the primary role and old primary becomes new standby.	Follow the generic switchover procedure described earlier to switch Streams Source.

Streams Destination Failover to DR site		

8	Start the propagation process on the new source. If the propagation has been dropped then it needs to be re-created.	Login as stream administrator on the source database and start the propagation process as follows:
		
		connect STRMADMIN/<Password>
		exec dbms_propagation_adm.start_propagation
		(propagation_name => '<Propagation Name>');
		
		If the propagation has been dropped then it needs to be recreated as follows:
		BEGIN
		dbms streams adm.add table_propagation_rules(
		table_name => '<SCHEMA>.<TABLE NAME>',
		streams_name => '<Stream Name>',
		source_queue_name =>
		'<Stream Administrators <Source Queue Name>',
		destination_queue_name =>
		'<Stream Administrators destination Queue Name>@<DBLINK>',
		include dml =>TRUE,
		include_ddl => FALSE,
		source_database => '<Source Database>',
		inclusion_rule => TRUE,
		queue to queue => TRUE);
		END;

9	Start Capture process on the new source database.	Login as stream administrator on the source database and start the capture process as follows: connect STRMADMIN/<Password> EXEC dbms_capture_adm.start_capture (capture_name => '<Capture Name>');

Streams Destination Failover to DR site		

10	Instantiate the object if necessary	connect STRMADMIN/*password*@SRC
		
		DECLARE
		  iscn NUMBER; — Variable to hold instantiation SCN value
		BEGIN
		  iscn :=
		DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
		  DBMS_DUTPUT.PUT_LINE ('Instantiation SCN is:' || iscn);
		
		-- Instantiate the objects at the destination database with this SCN value.
		
		DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@TGT
		(
		  source_object_name => '<SCHEMA>.<TABLE NAME>',
		  source_database_name => '<Source Database Name>',
		  instantiation_scn => iscn
		);
		END;

A13 PROCEDURE TO RECOVER FROM A STREAMS SOURCE FAILURE

This procedure provides guide lines as to how to deal with a situation of any accidental outage or catastrophic failure of a Streams source database. To deal with this situation if a business decision requires a role transitions, then we need to make sure that all streams source and destination are moved over to the DR site using the appropriate switchover and failover procedures.

This situation needs the following action:

a) Perform failover operation on Streams source database.

b) Perform switchover operation on Streams destination database.

A13.1 Perform the following steps to recover a Streams source failure.

Stream Source failover		
1	Perform a failover operation on the streams source database. Note: After a successful failover, the streams source standby database transitions to the primary and old primary becomes new standby	Follow the generic failover procedure to switch Streams Source.
2	Open the new primary Streams source database in restricted mode and make note of the point in time recovery SCN. **STANDBY BECAME PRIMARY SCN: _____**	Open up database in restricted mode startup restrict; Determine point in time recovery SCN SELECT standby_became_primary_scn FROM v$database;
3	On the new Streams source database, stop the capture process if already running.	Login as streams administrator on the source database connect STRMADMIN/<Password> Determine the status of capture process as follows: SELECT capture_name,status FROM dba_capture; If capture process is already running then stop the capture process as follows: EXEC dbms_capture_adm.stop_capture (capture_name => '<Capture Name'>);

4	Build LogMiner data dictionary for the capture process on the new source database. This process will output the lowest SCN value corresponding to the data dictionary extracted to the redo log.	set serveroutput on DECLARE
		l_scn NUMBER; BEGIN
		dbms_capture_adm.build(first_scn => l_scn);
		dbms_output.put_line('First SCN Value = ' || l_scn);
		END;

Stream Source failover		
5	At the Streams destination database, wait until all of the transactions from the source database in the apply process queue have been applied. The apply processes should become idle when these transactions have been applied. Note: The state should be IDLE for the apply process in both views before you continue.	Login as streams administrator on the destination database connect STRMADMIN/<Password> Verify the state of the apply process and wait until it becomes IDLE. SELECT apply_name, state FROM v$streams_apply_server; SELECT apply_name, state FROM v$streams apply reader;
6	Perform a query on the Streams destination database to determine the highest SCN for a transaction that was applied and note down this value. To obtain this value first find out the state of the apply process. If apply process is up and running then get this value from the V$STREAMS_APPLY_COORDINATOR view otherwise obtain this value from DBA_APPLY_PROGRESS view. HIGHEST APPLIED SCN ON DESTINATION:	If the apply process is already running, then perform the following query: SELECT hwm_message_number FROM v$streams_apply_coordinator WHERE apply_name = ' <Apply Name>'; If the apply process is disabled, then perform the following query: SELECT applied_message_number FROM dba_apply_progress WHERE apply_name = '<Apply Name>';
7	Destination is behind the Source scenario: If the HIGHEST APPLIED SCN ON DESTINATION noted in step 6 is less than the STANDBY_BECAME_PRIMARY_SCN noted in step 2 then the apply process on the streams destination database has not applied any transactions from the source database after STANDBY BECAME PRIMARY SCN.	Follow the recommended steps to deal with this situation: IF HIGHEST APPLIED SCN ON DESTINATION <= STANDBY_BECAME_PRIMARY_SCN Then • Stop the capture process on the new Streams source. EXEC dbms_capture_adm.stop_capture (capture_name => '<Capture Name>'); . Reset the start SCN for the capture process to the value obtained in step-6 on the Streams source database. EXEC dbms_capture_adm.alter_capture (capture_name => '<Capture Name>', start_scn => < HIGHEST APPLIED SCN ON DESTINATION>);__

Stream Source failover		
8	Source is behind the Destination scenario: If the HIGHEST APPLIED SCN ON DESTINATION obtained in step 6 is greater than or equal to the STANDBY BECAME PRIMARY SCN noted in step 2 then the apply process has applied transactions from the source database beyond what the new source database has.	Follow the recommended steps to deal with this situation: IF HIGHEST APPLIED SCN ON DESTINATION > STANDBY BECAME PRIMARY SCN Then Depending on the situation consider the following option: • Use flash back to flash back the Streams destination database to STANDBY BECAME PRIMARY SCN OR • Export the objects being streamed from the source database to the destination. Once the source and destinations are in sync then
9	Start the capture on the new source database as the Streams administrator.	exec dbms_capture_adm.start_capture ('<Capture Name'>);
10	Perform a switchover process on the Streams destination database.	Follow the generic switchover procedure to switch Streams destination. After a successful switchover, the streams destination standby database transitions to the primary role and old primary becomes new standby database.