

Implementing Heterogeneous Database Change Capture Solutions

March 2004

Implementing Heterogeneous Change Capture Solutions

Sharing information between non-Oracle and Oracle databases requires a custom solution. Streams can apply changes from Oracle to non-Oracle databases such as DB2, SQL Server, and SYBASE via a database link using a Transparent Gateway using the standard Streams apply process. However, many customers also need to capture changes made to data in non-Oracle databases, and send this data to the Oracle environment. Oracle does not provide heterogeneous database change capture functionality directly, but does provide standard interfaces to its Streams information distribution architecture, which will take any enqueued data changes, and disseminate them throughout the Oracle environment. This paper describes the basic requirements and techniques for implementing a customized change capture solution.

Requirements For Heterogeneous Database Change Capture

In this section, the requirements for a consultant or developer to provide Change Capture (CC) from a Non-Oracle System (NOS) into the Oracle Streams framework are outlined. The goal is to instantiate and maintain data from legacy systems to an Oracle grid environment.

The first sections are taken directly from the Streams documentation and describe a "custom application" that does CC from a NOS. For NOS CC to work with Oracle Streams, the developer must implement this custom application. These sections are:

- Non-Oracle to Oracle Data Sharing with 9.2 and 10.1 Streams
- Change Capture and Staging in a Non-Oracle to an Oracle 9.2 or 10.1 Environment
- Change Apply in a Non-Oracle to an Oracle 9.2 or 10.1 Environment
- Instantiation from a Non-Oracle Database to an Oracle Database

The middle section introduces the techniques that can be used for integrating the heterogeneous database change capture solution with Streams. This section describes the methods available to the vendor to integrate the captured non-Oracle change into the Streams queue.

The final sections of this document include a consolidated list of the ideal heterogeneous database change capture solution capabilities and the areas of focus for validation of the custom solution.

Non-Oracle to Oracle Data Sharing with 9.2 and 10.1 Streams

To capture and propagate changes from a non-Oracle database to an Oracle database, a custom application is required. This application gets the changes made to the non-Oracle database by reading from transaction logs, by using triggers, or by some other method. The application must assemble and order the

transactions and must convert each change into a logical change record (LCR). Then, the application must enqueue the LCRs into a queue in an Oracle database using either OCI AQ EnqArray, the DBMS_AQ procedures to enqueue messages, or JMS. The application must commit after enqueueing all LCRs in each transaction. Figure 1 shows a non-Oracle databases sharing data with an Oracle database.

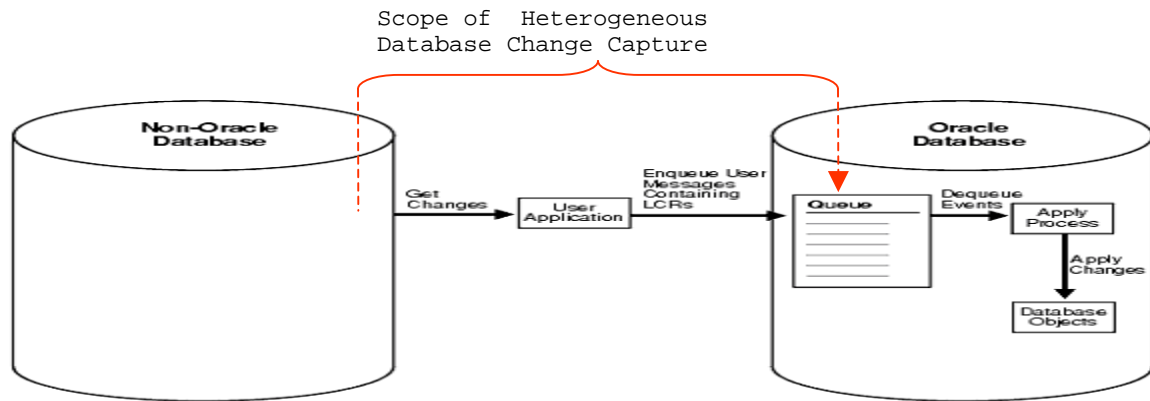


Figure 1: Non-Oracle to Oracle Heterogeneous Data Sharing

Change Capture and Staging in a Non-Oracle to an Oracle 9.2 or 10.1 Environment

Because the custom application is responsible for assembling changes at the non-Oracle database into LCRs and enqueueing the LCRs into a queue at the Oracle database, the application is completely responsible for change capture. This means that the application must construct LCRs that represent only committed changes at the non-Oracle database and then enqueue these LCRs into the queue at the Oracle database. The application must enqueue transactions serially in the same order as the transactions committed on the non-Oracle source database.

To ensure the same transactional consistency at both the Oracle database where changes are applied and the non-Oracle database where changes originate, a transactional queue must be used to stage the LCRs at the Oracle database. For example, suppose a single transaction contains three row changes, and the custom application enqueues three row LCRs, one for each change, and then commits. With a transactional queue, a commit is performed by the apply process after the third row LCR, retaining the consistency of the transaction. If a nontransactional queue is used, then a commit is performed for each row LCR by the apply process. The SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package creates a transactional queue automatically.

Change Apply in a Non-Oracle to an Oracle 9.2 or 10.1 Environment

In a non-Oracle to Oracle environment, the Streams apply process functions the same way as it would in an Oracle-only environment. That is, it dequeues each event from its associated queue based on apply process rules, performs any rule-based transformation, and either sends the event to a handler or applies it directly. Error handling and conflict resolution also function the same as they would in an Oracle-only environment. So, you can specify a prebuilt update conflict handler or create a custom conflict handler to resolve conflicts.

The apply process is an Oracle background process that dequeues events from a queue and either applies each event directly to a database object or passes the event as a parameter to a user-defined procedure called an apply handler. These apply handlers can include message handlers, DML handlers, and DDL handlers. You use rules to specify which events in the queue are applied.

For heterogeneous database change capture, the vendor supplied solution explicitly enqueues events into a queue as a Streams LCR for processing, enabling an apply process to apply them at a destination database. Alternatively, these events can be formatted as user messages that are processed by callbacks from an apply process. Figure 2 shows an apply process processing LCRs and user messages.

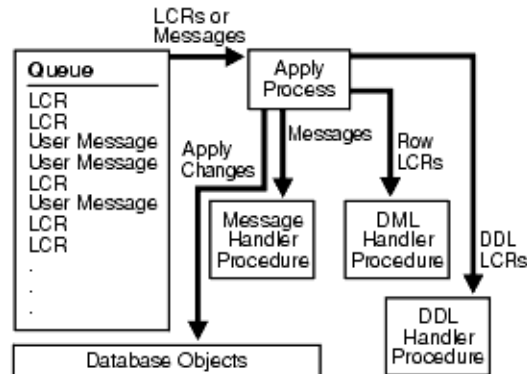


Figure 2: The Apply Process

Instantiation from a Non-Oracle Database to an Oracle Database

There is no automatic way to instantiate tables that exist at a non-Oracle database at an Oracle database. However, you can perform the following general procedure to instantiate a table manually:

1. At the non-Oracle database, use a non-Oracle utility to export the table to a flat file.
2. At the Oracle database, create an empty table that matches the table at the non-Oracle database.
3. At the Oracle database, use SQL*Loader to load the contents of the flat file into the table.

Note that developer would be responsible for guaranteeing that there are no lost transactions between the time of the NOS export and the enqueueing of NOS captured changes.

Instantiation is an integral part of the heterogeneous change data capture solution. It must be possible for users to continue actively working on the non-Oracle tables while the instantiation (export from non-Oracle/import to Oracle) itself is taking place using this solution, and changes made during the instantiation time period must be able to be captured, enqueued, and applied to the Oracle database without incurring errors due to data integrity issues.

Integration Techniques

Streams defines an XML schema for its Logical change record (LCR). Oracle also supports multiple techniques for enqueueing these XML LCR messages into the Streams queues, namely: OCI, PL/SQL, and JMS. Some third party vendors also define an XML Schema as a format for customers to access their "logical change record". If there is a mapping of one of the LCR schema to the other, it may

be trivial for the developer to quickly interoperate their existing change capture support from non-Oracle systems to Streams. The following sections discuss 9.2 and 10.1 integration:

- XML schema
- OCI support
- PL/SQL support
- JMS support
- Appendix A: Definition of the XML Schema for LCRs
- Appendix B: Example Oracle Streams configuration
- Appendix C: OCI and PL/SQL Enqueue Information
- Appendix D: Example Java Code for Enqueueing LCRs
- Appendix E: Documentation List

XML

Oracle Streams defines an XML schema for its LCR format. Some third party vendors of heterogeneous database change capture also publish their own XML schemas for changes. If the XML schemas are similar enough, it could be a straight forward process to use XML libraries to map the vendor's LCR to the Streams LCR. The definitions of the Streams LCR XML schema is in Appendix A. Currently the Streams LCR is based on a relational table. The application developer is responsible for the translation of data from non-relational systems to the Streams LCR format.

If the mapping between the Vendor LCR and the Streams LCR is not straight-forward in some cases, the developer can define their own message and enqueue it in the same transaction as the other LCRs. The developer can then define their own message handler that defines how the apply engine should process this message for this complex mapping. This is more complex approach and is not the recommended implementation.

OCI support (preferred method)

Oracle Call Interface (OCI), in combination with the XML LCR definition, is the preferred method of enqueueing the non-Oracle changes into the Oracle database. A complete set of transactional capabilities is supported with OCI, providing a robust toolset for application developers to manage the captured change processing between the heterogeneous systems.

There is no client-side definition of a Streams LCR for OCI. Instead, `OCIAQEnq` or `OCIAQEnqArray` can be used directly to enqueue XML LCRs. Upon enqueue, the XML LCRs will be automatically converted into the internal LCR representation used by Streams. `OCIAQEnq` is used to enqueue a single XML LCR into the Streams queue. To identify the end of a transaction, the `OCITransCommit` function is called. `OCIAQEnq` is available for both 9iR2 and 10g.

Further performance optimization is available in 10g using the `OCIAQEnqArray` function. With this function, the LCRs for a single transaction can be batched into an array and enqueued as a single operation. This is the preferred function for use in 10g implementations. Appendix C contains information pertaining to the use of `OCIAQEnqArray` and `OCIAQEnq`.

PL/SQL support

It is also possible to use OCI to invoke PL/SQL to enqueue a single LCR or an array of LCRs. The PL/SQL enqueue procedures enqueue explicitly constructed LCRs, not XML LCRs. To use the DBMS_AQ.ENQUEUE and ENQUEUE_ARRAY procedures, see the example on [Constructing and Enqueuing LCRs](#) in the Streams Replication Administrator's Guide, Chapter 9 "Managing LCRs". DBMS_AQ.ARRAY_ENQUEUE is available only in the Oracle 10g release. DBMS_AQ.ENQUEUE is supported in both the 9iR2 and 10g releases of the Oracle database.

JMS support

Sample code for enqueueing an LCR and a user message with JMS is provided in Appendix D. While JMS Streams support is available in both 9.2 and 10.1, it is the least desirable of the techniques listed due to performance limitations.

An Ideal Heterogeneous Database Change Capture Product

The following is a list of capabilities to include when developing an Heterogeneous Database Change Capture solution in approximate priority order.

- Minimal impact on the source system:
 - Log-based change capture is normally considered the least intrusive by customers
 - Little or no persistent storage on the source system
- Translation from a non-relational model to Streams-like LCR format
- Supports bulk instantiation with well-understood semantics
- Well defined recovery semantics that guarantees no loss of messages or transactions
- Guaranteed no lost transactions between instantiation and the start of change capture from the heterogeneous database
- Supports an XML access to the row-level changes
- Preserves transactional consistency
- Performs transaction assembly of changes and can deliver the transactions in non-interleaved commit order
- Can perform efficient flow control of capture

Validation of Heterogeneous Database Change Capture Solution Implementations

Listed below are the key areas for validation of a customized change capture solution.

Instantiation

- Ability to extract changes from non-Oracle database and get them into an Oracle database while users actively working on non-Oracle database tables (from which changes are being extracted).
- Error-free change capture immediately following instantiation
- Source and target tables are consistent after instantiation

Transaction Semantics

- Transactional integrity (transaction boundaries , batching of changes, etc)
- Non-Relational to Relational Mapping ability
- DDL support, if any

Message Management / Recovery

- No lost changes when non-Oracle or Oracle database become unavailable
- No duplicate changes enqueued into the Oracle database from the non-Oracle database due to instantiation timing OR database unavailability
- Exactly once delivery of changes to the Oracle Streams queue
- Source and target tables are consistent in a quiescent system despite failure scenarios

Appendix A: Definition of the XML Schema for LCRs

The XML schema described in this appendix defines the format of a logical change record (LCR). [Definition of the XML Schema for LCRs](#). This definition is valid for both 9.2 (9.2.0.5 and above) and 10.1. The highlighted portions of the XML definition are the 10.1 extensions.

The namespace for this schema is the following:
`http://xmlns.oracle.com/streams/schemas/lcr`

The schema is the following:
`http://xmlns.oracle.com/streams/schemas/lcr/streamslcr.xsd`

This schema definition can be loaded into the database by connecting as SYS in SQL*Plus and executing the following file:
`rdbms/admin/catxlcr.sql`

The rdbms directory is in your Oracle home.

Definition of the XML Schema for LCRs

The following is the XML schema definition for LCRs:

```
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xmlns.oracle.com/streams/schemas/lcr"
  xmlns:lcr="http://xmlns.oracle.com/streams/schemas/lcr"
  xmlns:xdb="http://xmlns.oracle.com/xdb"
  version="1.0"
  elementFormDefault="qualified">

  <simpleType name = "short_name">
    <restriction base = "string">
      <maxLength value="30"/>
    </restriction>
  </simpleType>

  <simpleType name = "long_name">
    <restriction base = "string">
      <maxLength value="4000"/>
    </restriction>
  </simpleType>

  <simpleType name = "db_name">
    <restriction base = "string">
      <maxLength value="128"/>
    </restriction>
  </simpleType>

  <!-- Default session parameter is used if format is not specified -->
  <complexType name="datetime_format">
    <sequence>
      <element name = "value" type = "string" nillable="true"/>
      <element name = "format" type = "string" minOccurs="0"
nillable="true"/>
    </sequence>
  </complexType>

  <complexType name="anydata">
    <choice>
      <element name="varchar2" type = "string" xdb:SQLType="CLOB"
nillable="true"/>
```



```

<!-- Represent char as varchar2. xdb:CHAR blank pads upto 2000 bytes! -->
<element name="char" type = "string" xdb:SQLType="CLOB"
                                nillable="true"/>
<element name="nchar" type = "string" xdb:SQLType="NCLOB"
                                nillable="true"/>

<element name="nvarchar2" type = "string" xdb:SQLType="NCLOB"
                                nillable="true"/>
<element name="number" type = "double" xdb:SQLType="NUMBER"
                                nillable="true"/>
<element name="raw" type = "hexBinary" xdb:SQLType="BLOB"
                                nillable="true"/>
<element name="date" type = "lcr:datetime_format"/>
<element name="timestamp" type = "lcr:datetime_format"/>
<element name="timestamp_tz" type = "lcr:datetime_format"/>
<element name="timestamp_ltz" type = "lcr:datetime_format"/>

<!-- Interval YM should be as per format allowed by SQL -->
<element name="interval_ym" type = "string" nillable="true"/>

<!-- Interval DS should be as per format allowed by SQL -->
<element name="interval_ds" type = "string" nillable="true"/>
<element name="urowid" type = "string" xdb:SQLType="VARCHAR2"
                                nillable="true"/>
</choice>
</complexType>

<complexType name="column_value">
  <sequence>
    <element name = "column_name" type = "lcr:long_name" nillable="false"/>
    <element name = "data" type = "lcr:anydata" nillable="false"/>
    <element name = "lob_information" type = "string" minOccurs="0"

nillable="true"/>
    <element name = "lob_offset" type = "nonNegativeInteger"
minOccurs="0"

nillable="true"/>
    <element name = "lob_operation_size" type = "nonNegativeInteger"
                                minOccurs="0"
nillable="true"/>
    <element name = "long_information" type = "string" minOccurs="0"

nillable="true"/>
  </sequence>
</complexType>

<complexType name="extra_attribute">
  <sequence>
    <element name = "attribute_name" type = "lcr:short_name"/>
    <element name = "attribute_value" type = "lcr:anydata"/>
  </sequence>
</complexType>

<element name = "ROW_LCR">
  <complexType>
    <sequence>
      <element name = "source_database_name" type = "lcr:db_name"

nillable="false"/>
      <element name = "command_type" type = "string" nillable="false"/>
      <element name = "object_owner" type = "lcr:short_name"

```

```

nillable="false"/>
    <element name = "object_name" type = "lcr:short_name"

nillable="false"/>
    <element name = "tag" type = "hexBinary" xdb:SQLType="RAW"
minOccurs="0"

nillable="true"/>
    <element name = "transaction_id" type = "string" minOccurs="0"
nillable="true"/>
    <element name = "scn" type = "double" xdb:SQLType="NUMBER"
minOccurs="0"

nillable="true"/>
    <element name = "old_values" minOccurs = "0">
        <complexType>
            <sequence>
                <element name = "old_value" type="lcr:column_value"
maxOccurs = "unbounded"/>

            </sequence>
        </complexType>
    </element>
    <element name = "new_values" minOccurs = "0">
        <complexType>
            <sequence>
                <element name = "new_value" type="lcr:column_value"
maxOccurs = "unbounded"/>

            </sequence>
        </complexType>
    </element>
    <element name = "extra_attribute_values" minOccurs = "0">
        <complexType>
            <sequence>
                <element name = "extra_attribute_value"
type="lcr:extra_attribute"
maxOccurs = "unbounded"/>

            </sequence>
        </complexType>
    </element>
    </sequence>
</complexType>
</element>

<element name = "DDL_LCR">
    <complexType>
        <sequence>
            <element name = "source_database_name" type = "lcr:db_name"
nillable="false"/>

            <element name = "command_type" type = "string" nillable="false"/>
            <element name = "current_schema" type = "lcr:short_name"
nillable="false"/>

            <element name = "ddl_text" type = "string" xdb:SQLType="CLOB"
nillable="false"/>

            <element name = "object_type" type = "string"
minOccurs = "0" nillable="true"/>
            <element name = "object_owner" type = "lcr:short_name"
minOccurs = "0" nillable="true"/>
            <element name = "object_name" type = "lcr:short_name"
minOccurs = "0" nillable="true"/>
            <element name = "logon_user" type = "lcr:short_name"
minOccurs = "0" nillable="true"/>
            <element name = "base_table_owner" type = "lcr:short_name"

```

```

minOccurs = "0" nillable="true"/>
<element name = "base_table_name" type = "lcr:short_name"
minOccurs = "0" nillable="true"/>
<element name = "tag" type = "hexBinary" xdb:SQLType="RAW"
minOccurs = "0" nillable="true"/>
<element name = "transaction_id" type = "string"
minOccurs = "0" nillable="true"/>
<element name = "scn" type = "double" xdb:SQLType="NUMBER"
minOccurs = "0" nillable="true"/>
<element name = "extra_attribute_values" minOccurs = "0">
  <complexType>
    <sequence>
      <element name = "extra_attribute_value"
type="lcr:extra_attribute"
maxOccurs = "unbounded"/>
    </sequence>
  </complexType>
</element>
</sequence>
</complexType>
</element>
</schema>' ;

```

Appendix B: Example Oracle Streams Configuration

This appendix is an example of configuring Oracle Streams in 9.2 to manipulate messages with OCI or JMS. It is assumed that Oracle 9.2.0.5 or above has been installed successfully, including XDB, before performing the steps in this example.

Step 1: Load XML Definitions for LCRs

The LCR schema must be loaded into the SYS schema using the catxldr.sql script in <ORACLE_HOME>\rdbms\admin directory. Run this script now if it has not been run already. Login to SQL* Plus as SYS user and execute this script as follows:

```
SQL>CONNECT / AS SYSDBA
```

```
SQL>@<ORACLE_HOME>/rdbms/admin/catxldr.sql
```

Execute this script on each database in which XML LCRs will be enqueued or dequeued in the Streams environment

Step 2: Init.ora Parameters

Verify that the init.ora parameter aq_tm_processes is set to a positive non-zero number. This is used by the queuing software for managing the cleanup of the queues after Streams has applied the enqueued messages.

Step3: Create Streams Administrator User

```
/*
```

Create the Streams administrator named strmadmin and grant this user the necessary privileges. In this example, the Streams administrator is also the apply user for the apply process and must be able to apply changes to the hr.emp_del table. Therefore, the Streams administrator is granted ALL privileges on this table.

```
*/
```

```
GRANT CONNECT, RESOURCE, DBA, SELECT_CATALOG_ROLE  
  TO strmadmin IDENTIFIED BY strmadmin;
```

```
GRANT ALL ON hr.emp_del TO strmadmin;
```

```
-- FOR 9.2 Databases do:
```

```
GRANT EXECUTE ON DBMS_APPLY_ADM          TO strmadmin;  
GRANT EXECUTE ON DBMS_AQ                 TO strmadmin;  
GRANT EXECUTE ON DBMS_AQADM             TO strmadmin;  
GRANT EXECUTE ON DBMS_STREAMS_ADM        TO strmadmin;
```

```
BEGIN
```

```
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(  
    privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,  
    grantee   => 'strmadmin',  
    grant_option => FALSE);
```

```
END;
```

```
/
```

```

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee   => 'strmadmin',
    grant_option => FALSE);
END;
/

```

-- For 10G Databases do:

```

BEGIN
  DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee      => 'strmadmin',
    grant_privileges => true);
END;
/

```

Step 4: Create the queue

```

/* Connect to Oracle database as the strmadmin user. */

CONNECT strmadmin/strmadmin

/*

```

Run the SET_UP_QUEUE procedure to create a queue named streams_queue. This queue will function as the Streams queue by holding the changes enqueued from the non-Oracle database and that will be dequeued by an apply process.

Running the SET_UP_QUEUE procedure performs the following actions:
Creates a queue table named streams_queue_table. This queue table is owned by the Streams administrator (strmadmin) and uses the default storage of this user. Creates a queue named streams_queue owned by the Streams administrator (strmadmin). Starts the queue.

To enqueue messages with JMS into the queue, enable the queue table (default queue table name from SET_UP_QUEUE is STREAMS_QUEUE_TABLE) for JMS types, using the ENABLE_JMS_TYPES procedure from the DBMS_AQADM package.

```

*/

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();

/* The following command is needed only for JMS access to the queue */
EXEC DBMS_AQADM.ENABLE_JMS_TYPES('STREAMS_QUEUE_TABLE'); --JMS only

```

Step 5: Create and Start the Apply process

```

/* Create an apply process, named APPLY01, to apply the user-enqueued LCRs to
local tables. This example takes advantage of the defaults for the apply
parameters, apply_captured (false) and apply_tag ('00'). The apply_tag can be
set to alternative values, such as '33', '56' as required by the customer. The
apply_tag parameter can be modified by the customer using the
DBMS_APPLY_ADM.ALTER_APPLY procedure.
*/

```

```

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name=>'strmadmin.streams_queue',
    apply_name=>'APPLY01');

```

```
END;
/

/*  Start the apply process APPLY01' */

BEGIN
    DBMS_APPLY_ADM.START_APPLY(
        apply_name=>'APPLY01');

END;
/
```

Appendix C: OCI and PL/SQL Enqueue Information

OCI

OCI allows the creation of applications that use the native procedures or function calls of a third-generation language to access an Oracle database server and control all phases of SQL statement execution. OCI supports the datatypes, calling conventions, syntax, and semantics of C and C++, including an interface to Oracle's Streams Advanced Queuing (Streams AQ) feature. OCI applications typically require a full client installation from the Oracle database distribution. Beginning with 10g, a smaller footprint "Instant Client" is available for selected Operating Systems. Mainframe systems continue to require the full client installation and ORACLE_HOME configuration.

The documentation for OCI is found in the [Oracle Call Interface Programmer's Guide](#). Additional information about OCI is also available at the Oracle Technology Network (OTN) website at <http://otn.oracle.com/tech/oci/index.html>

Using OCI to connect to the Oracle database, it is possible to enqueue XML LCRs into the Streams database. The array enqueue function, OCIAQEnqArray, is available only in Oracle 10g. The OCI enqueue function, OCIAQEnq, is available for supporting Oracle9.2 implementations. An XML LCR must be converted to an ANYDATA object before calling the OCIAQEnq* procedures. The OCI Typecode, OCI_TYPECODE_OPAQUE, is used whenever converting XML LCRs between XMLTYPE and ANYDATA. OCI code fragments are given below showing usage of the OCI array enqueue function calls for an XML LCR into the Streams queue. As in any application, be sure to confirm the successful completion of each OCI function call and handle any exceptions that might occur.

```
#define TYPE      "XMLTYPE"
#define TYPE1     "ANYDATA"
#define NUMMSGPERTXN (100)

OCIEnv          *envhp;
OCIServer       *srvhp;
OCIError        *errhp;
OCISvcCtx       *svchp;
OCIStmt        *stmthp;
OCISession      *usrhp;
OCIAQEnqOptions *enqopt;
OCIAQDeqOptions *deqopt;
OCIAQMsgProperties *msgprop_array_enq[NUMMSGPERTXN];
OCIXMLType      *xmllcr[NUMMSGPERTXN];
OCIInd          *xmllcr_ind[NUMMSGPERTXN];
OCIAnyData      *payload[NUMMSGPERTXN];
OCIInd          *payload_ind[NUMMSGPERTXN];
OCIInd          *payload_indp[NUMMSGPERTXN];
ub4             nummsgs;

/*-----
PRIVATE TYPES AND CONSTANTS
-----*/

Connect information based on Streams Configuration from Appendix B.
*/

static text *username = (text *) "strmadmin";
```

```

static text *password = (text *) "strmadmin";
static text *qname_enq = (text *) "STREAMS_QUEUE";
static text *sender = (text *) "strmadmin";
for ( i=0; i<nummsg; i++ {

/*   For each message, Convert OPAQUE type (xmllcr) to Anydata
(payload) */
OCIAnyDataConvert(svchp,
                  errhp, OCI_TYPECODE_OPAQUE,
                  xml_tdo, OCI_DURATION_SESSION, (dvoid *)nind,
                  xmllcr[i], 0, &payload[i]);
payload_ind[i]=OCI_IND_NOT_NULL;
payload_indp[i]=&payload_ind[i];
}
.....

/*   Use array enqueue to enqueue into Streams queue*/
OCIAQEnqArray(svchp,   errhp,
              (text *)qname_enq,
              enqopt, &iters,
              (OCIAQMsgProperties **) msgprop_array_enq,
              any_tdo,
              (dvoid **)payload,
              (dvoid **)payload_indp, (OCIRaw **)0,
              (dvoid *)0, (OCICallbackAQEnq)0, 0);
.....

/*   Commit the transaction */

OCITransCommit(svchp,
               errhp, (ub4) 0));

```

The documentation for the OCIAQEnqArray function is included in the [Oracle Call Interface Programmer's Guide](#)

PL/SQL

This section describes enqueueing LCRs into the Streams queue using the PL/SQL procedures ENQUEUE (for a single LCR) or ENQUEUE_ARRAY (for an array of LCRs) in the DBMS_AQ package. These procedures are limited to constructed LCRs, that is, an XML LCR definition cannot be enqueued in this manner.

To enqueue a user message containing an LCR into a SYS.AnyData queue using PL/SQL, first create the LCR to be enqueued. You use the constructor for the SYS.LCR\$_ROW_RECORD type to create a row LCR, and you use the constructor for the SYS.LCR\$_DDL_RECORD type to create a DDL LCR. Then you use the SYS.AnyData.ConvertObject function to convert the LCR into SYS.AnyData payload and enqueue it using the DBMS_AQ.ENQUEUE procedure.

Documentation for the DBMS_AQ.ENQUEUE_ARRAY procedure can be found in the PL/SQL Supplied Packages Manual. The Oracle Streams Advanced Queuing documentation (Chapter 24) includes an example that shows how to [construct an LCR and use the ENQUEUE](#) (or ENQUEUE_ARRAY) procedure to enqueue the message with PL/SQL.

Appendix D: Java Code for Enqueuing LCRs

This program uses the Oracle JMS API to publish non-LCR and LCR messages into a Streams topic. Streams allows both non-LCRs and LCRs to be managed by the same apply engine.

```
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;

public class StreamsEnq
{
    public static void main (String args [])
        throws java.sql.SQLException, ClassNotFoundException, JMSEException
    {
        TopicConnectionFactory tc_fact= null;
        TopicConnection        t_conn = null;
        TopicSession            t_sess = null;

        try
        {
            if (args.length < 3 )
                System.out.println("Usage:java filename [SID] [HOST] [PORT]");
            else
            {
                /* Create the TopicConnectionFactory
                 * Only the JDBC OCI driver can be used to access Streams through
JMS
                 */
                tc_fact = AQjmsFactory.getTopicConnectionFactory(
                    args[1], args[0], Integer.parseInt(args[2]),
"oci8");

                t_conn = tc_fact.createTopicConnection( "OE","OE");

                /* Create a Topic Session */
                t_sess = t_conn.createTopicSession(true,
Session.CLIENT_ACKNOWLEDGE);

                /* Start the connection */
                t_conn.start() ;

                /* Publish non-LCR based messages */
                publishUserMessages(t_sess);

                /* Publish LCR based messages */
                publishLcrMessages(t_sess);

                t_sess.close() ;
                t_conn.close() ;
                System.out.println("End of StreamsEnq Demo") ;
            }
        }
        catch (Exception ex)
        {
            System.out.println("Exception-1: " + ex);
            ex.printStackTrace();
        }
    }
}
```

```

    }

    /*
     * publishUserMessages - this method publishes an ADT message and a
     * JMS text message to a streams topic
     */
    public static void publishUserMessages(TopicSession t_sess) throws
Exception
    {
        Topic            topic            = null;
        TopicPublisher    t_pub            = null;
        JPerson           pers             = null;
        JAddress          addr             = null;
        TextMessage       t_msg            = null;
        AdtMessage        adt_msg          = null;
        AQjmsAgent        agent           = null;
        AQjmsAgent[]      recipList        = null;

        try
        {
            /* Get the topic */
            topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

            /* Create a publisher */
            t_pub = t_sess.createPublisher(topic);

            /* Agent to access oe_queue */
            agent = new AQjmsAgent("explicit_enq", null);

            /* Create a PERSON adt message */
            adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

            pers = new JPerson();
            addr = new JAddress();

            addr.setNum(new java.math.BigDecimal(500));
            addr.setStreet("Oracle Pkwy");

            pers.setName("Mark");
            pers.setHome(addr);

            /* Set the payload in the message */
            adt_msg.setAdtPayload(pers);

            ((AQjmsMessage)adt_msg).setSenderID(agent);

            System.out.println("Publish message 1 -type PERSON\n");

            /* Create the recipient list */
            recipList = new AQjmsAgent[1];
            recipList[0] = new AQjmsAgent("explicit_dq", null);

            /* Publish the message */
            ((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);

            t_sess.commit();

            t_msg = t_sess.createTextMessage();

            t_msg.setText("Test message");
            t_msg.setStringProperty("color", "BLUE");

```

```

        t_msg.setIntProperty("year", 1999);

        ((AQjmsMessage)t_msg).setSenderID(agent);

        System.out.println("Publish message 2 -type JMS TextMessage\n");

        /* Publish the message */
        ((AQjmsTopicPublisher)t_pub).publish(topic, t_msg, recipList);

        t_sess.commit();
    }
    catch (JMSException jms_ex)
    {
        System.out.println("JMS Exception: " + jms_ex);

        if(jms_ex.getLinkedException() != null)
            System.out.println("Linked Exception: " +
jms_ex.getLinkedException());
    }
}

/*
 * publishLcrMessages - this method publishes an XML LCR message to a
 * streams topic
 */
public static void publishLcrMessages(TopicSession t_sess) throws
Exception
{
    Topic                topic        = null;
    TopicPublisher        t_pub        = null;
    XMLType               xml_lcr      = null;
    AdtMessage            adt_msg       = null;
    AQjmsAgent            agent        = null;
    StringBuffer          lcr_data     = null;
    AQjmsAgent[]          recipList    = null;
    java.sql.Connection   db_conn      = null;

    try
    {
        /* Get the topic */
        topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

        /* Create a publisher */
        t_pub = t_sess.createPublisher(topic);

        /* Get the JDBC connection */
        db_conn = ((AQjmsSession)t_sess).getDBConnection();

        /* Agent to access oe_queue */
        agent = new AQjmsAgent("explicit_enq", null);

        /* Create a adt message */
        adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

        /* Create the LCR representation in XML */
        lcr_data = new StringBuffer();

        lcr_data.append("<ROW_LCR ");
        lcr_data.append("xmlns='http://xmlns.oracle.com/streams/schemas/lcr'
\n");

```

```

lcr_data.append("xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
\n");

lcr_data.append("xsi:schemaLocation='http://xmlns.oracle.com/streams/schemas/lc
r
http://xmlns.oracle.com/streams/schemas/lcr/streams_lcr.xsd'");
lcr_data.append("> \n");

lcr_data.append("<source_database_name>source_dbname</source_database_name>
\n");

lcr_data.append("<command_type>INSERT</command_type> \n");
lcr_data.append("<object_owner>Ram</object_owner> \n");
lcr_data.append("<object_name>Emp</object_name> \n");
lcr_data.append("<tag>0ABC</tag> \n");
lcr_data.append("<transaction_id>0.0.0</transaction_id> \n");
lcr_data.append("<scn>0</scn> \n");
lcr_data.append("<old_values> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C01</column_name> \n");
lcr_data.append("<data><varchar2>Clob old</varchar2></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C02</column_name> \n");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C03</column_name> \n");
lcr_data.append("<data> \n");
lcr_data.append("<date><value>1997-11-24</value><format>SYYYY-MM-
DD</format></date> \n");
lcr_data.append("</data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C04</column_name> \n");
lcr_data.append("<data> \n");

lcr_data.append("<timestamp><value>1999-05-
31T13:20:00.000</value><format>SYYYY-MM-DD'T'HH24:MI:
SS.FF</format></timestamp> \n");
lcr_data.append("</data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C05</column_name> \n");
lcr_data.append("<data><raw>ABCDE</raw></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("</old_values> \n");
lcr_data.append("<new_values> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C01</column_name> \n");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C02</column_name> \n");
lcr_data.append("<data><number>35.23</number></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C03</column_name> \n");
lcr_data.append("<data><number>-100000</number></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C04</column_name> \n");
lcr_data.append("<data><varchar>Hel lo</varchar></data> \n");

```

```

lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C05</column_name> \n");
lcr_data.append("<data><char>wor ld</char></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("</new_values> \n");
lcr_data.append("</ROW_LCR>");

/* Create the XMLType containing the LCR */
xml_lcr = oracle.xdb.XMLType.createXML(db_conn,
lcr_data.toString());

/* Set the payload in the message */
adt_msg.setAdtPayload(xml_lcr);

((AQjmsMessage)adt_msg).setSenderID(agent);

System.out.println("Publish message 3 - XMLType containing LCR
ROW\n");

/* Create the recipient list */
recipList = new AQjmsAgent[1];
recipList[0] = new AQjmsAgent("explicit_dq", null);

/* Publish the message */
((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);

t_sess.commit();

}
catch (JMSEException jms_ex)
{
    System.out.println("JMS Exception: " + jms_ex);

    if(jms_ex.getLinkedException() != null)
        System.out.println("Linked Exception: " +
jms_ex.getLinkedException());
    }
}
}

```

APPENDIX E: Documentation List

All Oracle documentation is available to the public at the Oracle Technology Network website (OTN). Access to the documentation does require a free membership on OTN. The URL for OTN is <http://otn.oracle.com/>

Additional information about both OCI and Streams is also available on OTN at <http://otn.oracle.com/products/dataint/content.html>

Oracle Software for most platforms is also available for evaluation download at the OTN website. Locate the DOWNLOADS service on OTN to acquire the Oracle 10g database.

Oracle Call Interface Documentation for Oracle 10g on OTN:

Oracle Call Interface Programmer's Guide	Detail information on programming with OCI	HTML	PDF
---	--	----------------------	---------------------

Oracle Streams Documentation for Oracle 10g on OTN:

Streams Concepts and Administration	Concepts and Administration of Oracle Streams. XML LCR type definition is in this book	HTML	PDF
Streams Replication Administrator's Guide	Detail information on Streams replication topics	HTML	PDF
Streams Advanced Queuing Java API Reference (Javadoc)		HTML	
Streams Advanced Queuing User's Guide and Reference	Streams Advanced Queuing documentation, includes info on DBMS_AQ.ENQUEUE and ENQUEUE_ARRAY as well as information for OCIEnqArray	HTML	PDF
PL/SQL Packages and Types Reference	All PL/SQL packages for Streams are documented here	HTML	PDF
Reference	All Oracle Views are documented in this book	HTML	PDF



Implementing Heterogeneous Database Change Capture Solutions
March 29, 2004

Author:

Contributing Authors: Alan Downing , Patricia McElroy, Bob Thome

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

www.oracle.com

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2004 Oracle Corporation

All rights reserved.