Informatica® Multidomain MDM
9.7.1 Hotfix 7 and later

Zero Downtime Upgrade Guide for Oracle

Informatica Multidomain MDM Zero Downtime Upgrade Guide for Oracle
9.7.1 Hotfix 7 and later
June 2020

Publication Date: 2020-06-30

Table of Contents

Preface

Follow the instructions in the Informatica® *Multidomain MDM Zero Downtime Upgrade Guide* to upgrade Multidomain MDM in a zero downtime environment. Zero Downtime is an optionally licensed feature that enables you to minimize disruptions while you upgrade Multidomain MDM.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Introduction to Zero Downtime

This chapter includes the following topics:

Zero Downtime Overview

Use Zero Downtime (ZDT) to upgrade Multidomain MDM while providing uninterrupted access to the MDM Hub. Batch and Services Integration Framework (SIF) user processes can run during the ZDT upgrade process.

To use ZDT, you have two environments: a passive environment and an active environment. You use Oracle GoldenGate to manage the data replication and message streams between the environments. While you upgrade the passive environment, MDM users can access the MDM Hub in the active environment. Any change made to the MDM Hub during the upgrade does not impact the ability to replicate changes from the active environment. Backfill mechanisms handle any impact to the MDM Hub metadata caused by the upgrade.

Oracle GoldenGate Processes for ZDT

ZDT requires a set of Oracle GoldenGate processes. Some of the processes extract and replicate data from the active environment to the passive environment. The remaining processes manage a bidirectional message stream. The messages reside in the C_REPOS_ZDT_EVENT_QUEUE repository table.

The following table identifies the prefixes that are used in the process group names:

Prefix	Process Type	Purpose
E_, P_	EXTRACT	Extracts data from the MDM Hub that is located in the active environment.
R_	REPLICAT	Replicates data to the MDM Hub that is located in the passive environment.

Prefix	Process Type	Purpose
EQ, PQ	EXTRACT	Extracts messages from the C_REPOS_ZDT_EVENT_QUEUE repository table. Runs on both environments to support a bidirectional message stream.
RQ	REPLICAT	Replicates messages to the C_REPOS_ZDT_EVENT_QUEUE repository table. Runs on both environments to support a bidirectional message stream.

For example, the following list shows the processes for ENVA and ENVB, where ENVA is the active environment.

```
GGSCI (hostname) 13> info all

Program     Status      Group
MANAGER     RUNNING
EXTRACT     RUNNING     EQENVA
EXTRACT     RUNNING     EQENVB
EXTRACT     RUNNING     E_ENVA
EXTRACT     RUNNING     PQENVA
EXTRACT     RUNNING     PQENVB
EXTRACT     RUNNING     P_ENVA
REPLICAT    RUNNING     RQENVA
REPLICAT    RUNNING     RQENVB
REPLICAT    RUNNING     R_ENVB
REPLICAT    RUNNING     R_ENVBU
```

Prerequisites for Zero Downtime

You must identity the source system and target system, ensure the database software is on both systems, install Oracle GoldenGate on both systems, and configure the MDM Hub Store databases for replication. For information about installing ZDT, see the *Multidomain MDM Zero Downtime Installation Guide for Oracle*.

Upgrade with Zero Downtime Workflow

When you upgrade with zero downtime, you perform the following general activities:

1. Upgrade the passive environment.
2. Switch the passive environment and the active environment.
3. Drop the former active environment.
4. Create the new passive environment from a copy of the new active environment.
5. Replicate all data changes that occur during the upgrade so that the environments are the same at the end of the upgrade process.

In this guide, some steps are conditional. Perform these steps only when you are doing a particular type of update or upgrade.

Conditional steps have one or more of the following prefixes:

- **Schema Update**. You changed an Operational Reference Store schema based on business requirements.
- **Schema Update with Data Change**. You changed an Operational Reference Store schema and some data in the database.
- **MDM Upgrade**. You are upgrading Multidomain MDM to a new major, minor, or hotfix release, or you are applying an emergency bug fix.
- **Infrastructure Upgrade**. You are upgrading other software or hardware in the same environments in which Multidomain MDM runs.

You can run the steps for the ZDT upgrade from a command line interface, such as shell scripts, or a command-line job scheduler. The upgrade runs from a single flow of control to allow for almost complete automation of the upgrade process. The ZDT upgrade procedure includes steps for messaging between the active environment and passive environment, maintaining replication control, and integrating backfill.

Upgrading Multidomain MDM from version 9.5.0 or earlier

In Multidomain MDM version 9.5.1, the data structure for the master database schema changed. If you are upgrading from version 9.5.0 or earlier, you must go through a readiness cycle before you start the upgrade cycle. The readiness cycle identifies data issues that you must resolve before you upgrade. After the readiness cycle, when you upgrade with ZDT, the upgrade cycle updates the data structure in the schema.

To request a version of this guide that contains the steps to upgrade from version 9.5.0 or earlier, contact Informatica Global Customer Support.

CHAPTER 2

Upgrading with Zero Downtime

This chapter includes the following topics:

Edit the Samples

This guide contains sample code and scripts. To use the samples, copy the samples and substitute your own values. The samples use the forward slash in paths. Ensure that you use the file path separator that works with your operating system.

Transfer the Previous Load Table Data

You can copy the data from the previous load table in the active environment to the previous load table in the passive environment. The data in the previous load table is not transferred during the replication process. If you do not replicate the previous load table, the first stage batch job that you run after the passive environment becomes active might process all the data in the landing table. The names of the previous load tables end in _PRL.

Note: Before you run a stage batch job, ensure that the upgrade is complete. If you run a stage job before the upgrade is complete, the data that the stage batch job adds to the source previous load tables is not added to the target previous load tables.

1. Generate the `prl_expdp.prm` and `prl_impdp.prm` parameter files for the previous load table in the active environment.

   From SQL*Plus in the active environment, run the following command:

```
DECLARE
   IN_TYPE NUMBER;
BEGIN
   IN_TYPE := 1;--generate prl_expdp.prm file
   CMXZDT.write_prl_exp_imp_param(IN_TYPE => IN_TYPE);
   IN_TYPE := 2; --generate prl_impdp.prm file
   CMXZDT.write_prl_exp_imp_param(IN_TYPE => IN_TYPE);
```

```
END;
/
```

The parameter files are generated in the `GGS/dirprm` directory.

2. Open a command prompt and navigate to the `GGS/dirprm` directory.
3. Export the data in the previous load table in the active environment:

```
expdp <active Operational Reference Store name>/<TNS password>@<TNS name>
parfile=prl_expdp.prm
```

The `prl.dmp` file is generated in `GGS/dirprm`.

4. Copy the `PRL.dmp` and `prl_impdp.prm` files from the `GGS/dirprm` directory in the active environment to the `GGS/dirprm` directory in the passive environment.
5. If the active and passive environments are not swapping for the first time, truncate the previous load table in the passive environment. You must truncate the previous load table in the passive environment so that the data from the active previous load table can be imported.

   In the passive environment, log in to SQL*Plus and run the following command:

```
TRUNCATE TABLE <Previous Load Table Name>;
```

6. In the passive environment, open a command prompt and navigate to `GGS/dirprm`.
7. Run the following command to import the data into the previous load table in the passive environment:

```
impdp <passive Operational Reference Store name>/<TNS password>@<TNS name>
remap_schema=<active Operational Reference Store name>:<passive Operational
Reference Store name> parfile=prl_impdp.prm
```

Upgrade Steps Controlled from the Passive Environment

To ensure the systems are synchronized, you must have replication running from the active environment to the passive environment.

Important: In the following procedure, every step is dependent on the previous steps. Run all the commands in order. Unless otherwise specified, ensure that each process is finished before you run the next command.

1. **MDM Upgrade**. Stop the application server.
2. Disable read services to the passive environment.
3. Connect to the Operational Reference Store schema that you are upgrading.
4. Set `ACTIVE_UPGRADE_IND` to `1` in the passive environment:

```
update C_REPOS_ZDT_STATUS set ACTIVE_UPGRADE_IND = 1;
BEGIN
   cmxzdt.v_zdt_ind:= -1;
END;
/
commit;
```

5. Turn off replication replay in the passive environment:

```
DECLARE
out_error_msg VARCHAR2(32000);
out_return_code INT;
BEGIN
   cmxzdt.stop_replication_replay(out_error_message => out_error_msg,
                                  out_return_code => out_return_code);
```

```
   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

6. Clean up schema changes from previous upgrades.

   The following commands remove all obsolete columns and tables from the passive environment based on C_REPOS_MET_VIRTUAL_CHANGE:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   cmxzdt.cleanup_obsolete_objects(out_error_message => out_error_msg,
                                   out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

7. **MDM Upgrade**. Stop all Oracle GoldenGate process in the passive environment.

   Warning: If Oracle GoldenGate processes are running during the upgrade, the upgrade process cannot recompile the `CMXZDT` packages. In this case, the `CMXZDT` packages become not valid due to schema changes.

```
DECLARE
    out_error_msg   VARCHAR2(32000);
    out_return_code INT;
BEGIN
    cmxzdt.configure_ggs_event_replicat( cmxzdt.zdt_local );
    cmxzdt.stop_event_queue( out_error_msg, out_return_code );

    dbms_output.put_line('out_return_code => ' || out_return_code || ' out_error_msg
=> ' || SUBSTR( out_error_msg, 1, 250 ));
END;
/
```

8. **MDM Upgrade**. Upgrade Multidomain MDM:

   1. Upgrade the Hub Store.
   2. Upgrade the Hub Server.
   3. Upgrade the Process Server.

   For more information about upgrading the software, see the *Multidomain MDM Upgrade Guide*.

   Important: After you finish the upgrade, the application server must be running so that you can perform the remaining steps.

9. **Schema Update**. Run the Metadata Manager command line utility to apply the changelist.

   The utility applies the changes in the changelist file to the schema. For example, you can use a changelist to add or remove columns in the base object table or to set the trust values on columns.

10. Run a full tokenize batch job to update all the match tokens in the *_STRP tables.

You can run the tokenize back job from the Hub Console or by using a service integration framework (SIF) API.

Run From	Steps
Hub Console	1. In the MDM Hub Console, open the Batch Viewer tool. 2. From the Batch Viewer navigation pane, expand the base object for which you want to regenerate all the match tokens. 3. Expand **Generate Match Tokens**. 4. Select the batch job that you want to use to generate match tokens. 5. Select **Re-generate All Match Tokens**. 6. Click **Execute Batch**.
API	To run a tokenize batch job on all records, use the ExecuteBatchGenerateMatchTokens request with the `fullRestripInd` set to `1`. The following code sample shows an ExecuteBatchGenerateMatchTokens request to create match tokens for all records in the C_PARTY base object: `<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:siperian.api"> <soapenv:Header/> <soapenv:Body> <urn:executeBatchGenerateMatchTokens> <urn:username>admin</urn:username> <urn:password> <urn:password>admin</urn:password> <urn:encrypted>false</urn:encrypted> </urn:password> <urn:orsId>localhost-orclsnl-UTSOURCE</urn:orsId> <urn:asynchronousOptions> <urn:isAsynchronous>false</urn:isAsynchronous> <urn:jmsReplyTo></urn:jmsReplyTo> <urn:jmsCorrelationId></urn:jmsCorrelationId> </urn:asynchronousOptions> <urn:tableName>C_PARTY</urn:tableName> <urn:fullRestripInd>1</urn:fullRestripInd> </urn:executeBatchGenerateMatchTokens> </soapenv:Body> </soapenv:Envelope>`

11. **MDM Upgrade** and **Schema Update**. Populate the backfill table C_REPOS_ZDT_BACKFILL_TASK to indicate that trust backfill is required on base object tables.

    Use the `CMXZDT.add_backfill_task()` method:

    ```
    exec CMXZDT.add_backfill_task(backfill type, 'base object name', 'api', sequence);
    COMMIT;
    ```

    where:

    - *backfill type* is the type of backfill:
      - `TRUST_BACKFILL`. **Recommended**. Use when you add new trusted columns. This option runs the same processes that are run by both the `REVALIDATE` and `RECALCULATE` options.
      - `REVALIDATE`. Use when you change or add validation rules.
      - `RECALCULATE`. Use when you change trust rules.
      - `TOKENIZE`. Use when you need to run the tokenize process on dirty records, but you are unable to run batch jobs.

- *base object name* is the table name of a base object. Run the command on all base objects tables where you want to recalculate the best version of the truth (BVT). If you are not sure which tables are affected by the schema update, run the command on all base object tables in the schema.
- *api* specifies which API runs the backfill task. `R` is Read API, `W` is Write API, and `B` is both Read and Write APIs. Use `B`.
- *sequence* is the order in which to run the backfill task in relationship to other tasks. If you are not sure, use `1` to run the backfill task first.

For example, the following command applies trust backfill for the C_CUSTOMER base object:

```
exec CMXZDT.add_backfill_task('TRUST_BACKFILL','C_CUSTOMER','B', 1);
COMMIT;
```

In the command, use the data in the repository metadata.

12. Run the backfill batch job for each base object.

    You can run the batch backfill job from the Hub Console or by using a service integration framework (SIF) API.

Run From	Steps
Hub Console	1. In the MDM Hub Console, open the Batch Viewer tool. 2. From the Batch Viewer navigation pane, select the base object that you want to backfill. If the backfill batch job does not appear in the batch viewer for the base object, select **Batch Viewer** > **Refresh**. 3. Run the backfill batch job.

API	1. Ensure the MDM Hub Server is running.
	2. Select an API.
	• To run a backfill on all base objects, use the ExecuteBatchBackfillAll API.
	• To run a backfill a specified base object, use the ExecuteBatchBackfill API.
	3. To run the backfill on all records, ensure that the `dirtyOnlyInd` parameter is `false`.
	4. **Important:** Comment out the rowidObjectTable element in the request.
	
	The following code sample shows an ExecuteBatchBackfill request to backfill records in the C_BO_TRUST base object:
	`<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:siperian.api">`
	`<soapenv:Header/>`
	`<soapenv:Body>`
	`<urn:ExecuteBatchBackfill>`
	`<urn:username><user name></urn:username>`
	`<urn:password>`
	`<urn:password><password></urn:password>`
	`</urn:password>`
	`<urn:orsId>localhost-orclsnl-UTSOURCE</urn:orsId>`
	`<urn:asynchronousOptions>`
	`<urn:isAsynchronous>false</urn:isAsynchronous>`
	`</urn:asynchronousOptions>`
	`<urn:tableName>C_BO_TRUST</urn:tableName>`
	`<!--urn:rowidObjectTable>?</urn:rowidObjectTable-->`
	`<urn:dirtyOnlyInd>`**`false`**`</urn:dirtyOnlyInd>`
	`</urn:ExecuteBatchBackfill>`
	`</soapenv:Body>`
	`</soapenv:Envelope>`

13. **Schema Update with Data Change**. Disable the regular Oracle GoldenGate mapping for the table C_AGREEMENT and change the mapping to table C_AGREEMENT_XREF_NEW_FROM_A.

    a. Turn off mapping for the entire C_AGREEMENT base object and all related tables by using cmxzdt.disable_bo_replication ('C_AGREEMENT'):

    ```
    BEGIN
       cmxzdt.disable_bo_replication ('C_AGREEMENT');
    commit;
    END;
    /
    ```

    b. Create a new mapping from C_AGREEMENT_XREF to C_AGREEMENT_XREF_NEW_FROM_A by using `cmxzdt.remap_table_replication ('C_AGREEMENT_XREF', ' C_AGREEMENT_XREF_NEW_FROM_A')`. The source and target tables must be the same. The table C_AGREEMENT_XREF_NEW_FROM_A is created automatically by cmxzdt.remap_table_replication. If the table exists, the procedure fails. Run the following command:

    ```
    BEGIN
       cmxzdt.remap_table_replication ('C_AGREEMENT_XREF',
    'C_AGREEMENT_XREF_NEW_FROM_A');
    commit;
    END;
    /
    ```

14. **Schema Update with Data Change**. Start the reload of data to C_AGREEMENT from C_AGREEMENT_XREF. You do not need to wait for the data to reload before you proceed to the next step.

15. If you conduct user acceptance validation, perform the following steps:

    a. Note the system change number (SCN) for later flashback.

    b. Perform user acceptance validation.

    c. Flash back to the SCN noted in substep a.

16. **MDM Upgrade**. Start event queue processes in the passive environment:

    ```
    DECLARE
        out_error_msg   VARCHAR2(32000);
        out_return_code INT;
    BEGIN
        CMXZDT.CONFIGURE_GGS_EVENT_REPLICAT(CMXZDT.ZDT_LOCAL);
        cmxzdt.start_event_queue( out_error_msg, out_return_code );
        dbms_output.put_line('out_return_code => ' || out_return_code || ' out_error_msg
    => ' || SUBSTR( out_error_msg, 1, 250 ));
    END;
    /
    ```

17. Process data changes from the passive environment by starting replication replay:

    ```
    DECLARE
       out_error_msg VARCHAR2(32000);
       out_return_code INT;
    BEGIN
       cmxzdt.start_replication_replay(in_env_type => cmxzdt.zdt_local,
                                 out_error_message => out_error_msg,
                                   out_return_code => out_return_code);

       dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
    '||substr(out_error_msg,1,250));
    END;
    /
    ```

18. Detect completion of replication catch-up in the passive environment:

    ```
    DECLARE
        out_error_msg VARCHAR2(32000);
        out_return_code INT;
    BEGIN
    ```

```
    cmxzdt.wait_replay_catchup(in_timeout_minutes => 5,
                               out_error_message => out_error_msg,
                                 out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

19. Send a message from the passive environment to the active environment to disallow batch processes in the active environment:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   cmxzdt.disable_all_batch(in_affected_zdt_env => cmxzdt.zdt_source,
                           in_timeout_minutes => 30,
                            out_error_message => out_error_msg,
                              out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||'  out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

Changes would be required to the batch scheduling and batch control script to detect this condition and not attempt new jobs. This can be controlled based on the flag on environment A in C_REPOS_ZDT_STATUS.batch_disabled_ind. Any subsequent invocation of batch jobs from the active environment will result in an error. The job scheduler should be updated accordingly.

20. Enable writeable SIF services in the passive environment:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   cmxzdt.enable_all_write_sif(in_affected_zdt_env => cmxzdt.zdt_target,
                               in_timeout_minutes => 5,
                                out_error_message => out_error_msg,
                                  out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

21. Verify that the application server is running in the passive environment, and then redirect services from the active environment to the passive environment.

    The passive environment is live for read and write services.

22. Synchronize sequences.

    Sequences on the target system are not updated through replication. Reset the sequence numbers so that, after switching the passive and active environments, the new sequence values on the target system are higher than on the source system.

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   cmxzdt.synchronize_sequences(out_error_message => out_error_msg,
                                  out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

23. Disable writeable SIF services in the active environment:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   cmxzdt.disable_all_write_sif(in_affected_zdt_env => cmxzdt.zdt_source,
                                in_timeout_minutes => 5,
                                 out_error_message => out_error_msg,
                                   out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

24. Complete the replication processing in the passive environment:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   cmxzdt.stop_replication(out_error_message => out_error_msg,
                              out_return_code => out_return_code,
                           in_timeout_minutes => 60);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

This call returns a response when all data is replicated to the passive environment.

25. **Schema Update with Data Change**. Process the delta on the C_AGREEMENT table.

The implementation resource custom writes this to handle the data that came from the active environment after you reloaded the passive environment.

26. **MDM Upgrade** and **Schema Update**. Run the backfill job on dirty records for each base object.

Run From	Steps
Hub Console	1. In the MDM Hub Console, open the Batch Viewer tool. 2. From the Batch Viewer navigation pane, select the base object that you want to backfill. If the backfill batch job does not appear in the batch viewer for the base object, select **Batch Viewer** > **Refresh**. 3. To run the backfill on only the dirty records, select **For dirty records only**. 4. Run the backfill batch job.

Run From	Steps

API	1. Ensure the MDM Hub Server is running.
	2. Select an API.
	• To run a backfill on all base objects, use the ExecuteBatchBackfillAll API.
	• To run a backfill a specified base object, use the ExecuteBatchBackfill API.
	3. To run the backfill on only the dirty records, set the `dirtyOnlyInd` parameter to `true`.
	4. **Important:** Comment out the rowidObjectTable element in the request.
	
	The following code sample shows an ExecuteBatchBackfill request to backfill the dirty records in the C_BO_TRUST base object:
	`<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:siperian.api">`
	`<soapenv:Header/>`
	`<soapenv:Body>`
	`<urn:ExecuteBatchBackfill>`
	`<urn:username><user name></urn:username>`
	`<urn:password>`
	`<urn:password><password></urn:password>`
	`</urn:password>`
	`<urn:orsId>localhost-orclsnl-UTSOURCE</urn:orsId>`
	`<urn:asynchronousOptions>`
	`<urn:isAsynchronous>false</urn:isAsynchronous>`
	`</urn:asynchronousOptions>`
	`<urn:tableName>C_BO_TRUST</urn:tableName>`
	`<!--urn:rowidObjectTable>?</urn:rowidObjectTable-->`
	`<urn:dirtyOnlyInd>`**`true`**`</urn:dirtyOnlyInd>`
	`</urn:ExecuteBatchBackfill>`
	`</soapenv:Body>`
	`</soapenv:Envelope>`

27. Remove the backfill tasks from the C_REPOS_BACKFILL_TASK table. The table must be empty so that other batch jobs can run.

```
Delete from c_repos_zdt_backfill_task;
COMMIT;
```

28. **MDM Upgrade** and **Schema Update**. Run a tokenize batch job on dirty records for each base object.

Run From	Steps
Hub Console	1. In the MDM Hub Console, open the Batch Viewer tool. 2. From the Batch Viewer navigation pane, expand the base object for which you want to regenerate all the match tokens. 3. Expand **Generate Match Tokens**. 4. Select the batch job that you want to use to generate match tokens. 5. Clear **Re-generate All Match Tokens**. 6. Click **Execute Batch**.

Run From	Steps
API	To run the tokenize batch job on dirty records only, use the ExecuteBatchGenerateMatchTokens request with the `fullRestripInd` set to `0`. The following code sample shows an ExecuteBatchGenerateMatchTokens request to create match tokens for dirty records in the C_PARTY base object: `<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:siperian.api"> <soapenv:Header/> <soapenv:Body> <urn:executeBatchGenerateMatchTokens> <urn:username>admin</urn:username> <urn:password> <urn:password>admin</urn:password> <urn:encrypted>false</urn:encrypted> </urn:password> <urn:orsId>localhost-orclsnl-UTSOURCE</urn:orsId> <urn:asynchronousOptions> <urn:isAsynchronous>false</urn:isAsynchronous> <urn:jmsReplyTo></urn:jmsReplyTo> <urn:jmsCorrelationId></urn:jmsCorrelationId> </urn:asynchronousOptions> <urn:tableName>C_PARTY</urn:tableName> <urn:fullRestripInd>0</urn:fullRestripInd> </urn:executeBatchGenerateMatchTokens> </soapenv:Body> </soapenv:Envelope>`

29. Enable batch services in the passive environment:

```
Sample SQL:
DECLARE
    out_error_msg VARCHAR2(32000);
    out_return_code INT;
BEGIN
    cmxzdt.enable_all_batch(in_affected_zdt_env => cmxzdt.zdt_target,
                                  in_timeout_minutes => 5,
                                   out_error_message => out_error_msg,
                                     out_return_code => out_return_code);

    dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
END;
/
```

30. **Schema Update with Data Change**. If any user data foreign keys changed during the update, run foreign key validation. Run the ExecuteBatchValidateFKRelationships SIF API for each base object.

    Any violations that came from the active environment after the lookup data is updated are detected.

31. **Schema Update**. If there are violations, fix the violations.

    Tip: If the violations are not severe, you can fix the violations after you complete the upgrade. If you cannot fix some violations, contact Informatica Global Customer Support.

32. Stop all Oracle GoldenGate processes in the passive environment and the active environment. Clean the passive environment.

    a. From the GoldenGate command line, stop all GoldenGate processes in the passive environment and remove processes and trail files:

```
GGSCI (hostname) 1> dblogin userid USER password "Password"
GGSCI (hostname) 1> stop *
GGSCI (hostname) 1> delete * -- hit 'y' when Prompted for n/y
```

```
GGSCI (hostname) 1> DELETE TRACETABLE [SCHEMA_NAME].GGS_EVENT_TRACE
GGSCI (hostname) 1> DELETE CHECKPOINTTABLE [SCHEMA_NAME].GGS_EVENT_CHECKPOINT
GGSCI (hostname) 1> DELETE CHECKPOINTTABLE [SCHEMA_NAME].GGS_CHECKPOINT
```

b. Stop all processes in the active environment and remove processes and trail files:

```
GGSCI (hostname) 1> dblogin userid USER password "Password"
GGSCI (hostname) 1> stop *
GGSCI (hostname) 1> delete * -- hit 'y' when Prompted for n/y
GGSCI (hostname) 1> DELETE TRACETABLE [SCHEMA_NAME].GGS_EVENT_TRACE
GGSCI (hostname) 1> DELETE CHECKPOINTTABLE [SCHEMA_NAME].GGS_EVENT_CHECKPOINT
```

c. Delete all the files related to the passive environment. Navigate to each of the following directories and remove the specified files:

```
<GoldenGate install directory>/dirchk
del *

<GoldenGate install directory>/dirdat
del enva/*
del envb/*

<GoldenGate install directory>/dirprm
del eqenv*.prm*
del pqenv*.prm*
del rqenv*.prm*
del e_env*.prm*
del p_env*.prm*
del *.def

<GoldenGate install directory>/dirrpt
del *.dsc
del e_env*.rpt
del eqenv*.rpt
del p_env*.rpt
del pqenv*.rpt
del r_env*.rpt
del rqenv*.rpt
```

d. Repeat the preceding step on all the files related to the active environment.

e. Remove the history from the ZDT event queue in the passive environment.

Important: Do not drop the GGS_CHECKPOINT and GGS_CHECKPOINT_LOX tables. If you drop these tables, the REPLICAT service does not start.

```
/* Repository tables for ZDT */
delete from C_REPOS_ZDT_EVENT_QUEUE;
delete from C_REPOS_ZDT_REPLICAT_EXCEPTION;
update C_REPOS_ZDT_ENV_STATE set state = NULL, state_ts = NULL, state_desc =
NULL, updated_by=NULL, update_date=NULL;

/* Tables for Oracle GoldenGate */
delete from GGS_CHECKPOINT;        -- Do not drop this table
delete from GGS_CHECKPOINT_LOX;    -- Do not drop this table

drop table GGS_EVENT_CHECKPOINT cascade constraints;
drop table GGS_EVENT_CHECKPOINT_LOX cascade  constraints;
drop table GGS_EVENT_TRACE cascade  constraints;
commit;
```

f. Search for %GGS% tables. Verify that the GGS_CHECKPOINT and GGS_CHECKPOINT_LOX tables exist. If you find any other tables with GGS in the table name, including TMP_GGS_* tables, you can drop these other tables.

33. **MDM Upgrade**. Remove all temporary procedures and packages used for the backfills:

```
DECLARE

    PROCEDURE unregister_table(
      prowid_table VARCHAR2
```

```
    , ptable       VARCHAR2
    )
    AS
        sql_text VARCHAR2(2000);
    BEGIN

        debug_print(
          ' Start to unregister a table: ' || ptable ||
          ' rowid_table ' || prowid_table
        );

        FOR cc2 IN (
            SELECT k.rowid_key_constraint, k.key_type_str, kc.rowid_column
                 , kc.rowid_parent_column
              FROM c_repos_key_constraint k
                 , c_repos_key_constraint_col kc
                 , c_repos_column c
             WHERE k.rowid_key_constraint = kc.rowid_key_constraint
               AND kc.rowid_column = c.rowid_column
               AND c.rowid_table = prowid_table
        ) LOOP
            debug_print(
               ' Start to deprecate constraints : ' || cc2.rowid_key_constraint ||
               '  ' || cc2.rowid_column || ' rowid_table ' || prowid_table
            );

            sql_text :=
                'DELETE c_repos_key_constraint_col ' ||
                 'WHERE rowid_column = ''' || cc2.rowid_column || '''';
            debug_print( sql_text );
            EXECUTE IMMEDIATE sql_text;

            sql_text :=
                'DELETE c_repos_key_constraint ' ||
                 'WHERE rowid_key_constraint = ''' || cc2.rowid_key_constraint ||
''''';
            debug_print( sql_text );
            EXECUTE IMMEDIATE sql_text;

        END LOOP;

        sql_text :=
            'DELETE c_repos_column ' ||
             'WHERE rowid_table = ''' || prowid_table || '''';
        debug_print( sql_text );
        EXECUTE IMMEDIATE sql_text;

        sql_text :=
            'DELETE c_repos_table ' ||
             'WHERE rowid_table = ''' || prowid_table || '''';
        debug_print( sql_text );
        EXECUTE IMMEDIATE sql_text;

        COMMIT;
        debug_print(' unregistered table: ' || prowid_table);

    END unregister_table;

BEGIN
    debug_print(' ========================================== ');
    debug_print('==> Start to drop temporary ZDT objects');
    debug_print('==> ');
    FOR i IN (
        SELECT DISTINCT u.object_type t, u.object_name n
          FROM user_procedures u
         WHERE object_name IN ( 'ZDT_C_REPOS_TASK_DATA_GET_XREF'
                              , 'GENERATE_CMXZDT_TEMP'
                              , 'CMXZDT_TEMP' )
    ) LOOP
```

```
        debug_print( '==> Start to drop temporary ' || i.t || ' ' || i.n || '' );
        EXECUTE IMMEDIATE 'drop ' || i.t || ' ' || i.n || '';
    END LOOP;
    debug_print( '==> ' );
    debug_print( '==> End dropping temporary ZDT objects' );
    debug_print( ' ============================================== ' );

    debug_print( ' ============================================== ' );
    debug_print( '==> Start to unregister deprecated tables ' );
    debug_print( '==> ' );
    FOR cc IN (
        SELECT rowid_table, table_name
             , SUBSTR( table_name, LENGTH( table_name ) - 3, 4 ) AS postfix
             , SUBSTR( UPPER( table_name ), 1, LENGTH( table_name ) - 5 ) AS bo_name
                 FROM c_repos_table
                WHERE type_ind = 14
                  AND ( ( table_name LIKE '%_HUID' ) OR
                        ( table_name LIKE '%_HFKM' ) OR
                        ( table_name LIKE '%_HMXR' )
                      )
    ) LOOP
        debug_print(
          ' Start to unregister table : ' || cc.table_name ||
          ' rowid_table ' || cc.rowid_table
        );
        unregister_table( cc.rowid_table, cc.table_name );
    END LOOP;

    debug_print( '==> ' );
    debug_print( '==> End unregistering deprecated tables ' );
    debug_print( ' ============================================== ' );

    debug_print( 'Changes are completed for all tables. ' );

END;
/
```

34. Mark the passive environment as the source to set up for the replication switch:

```
update c_repos_zdt_status set REPLICATION_TARGET_IND = 0;
COMMIT;
```

35. In the C_REPOS_ZDT_STATUS table, set `ACTIVE_UPGRADE_IND` to `0` in the passive environment:

```
update C_REPOS_ZDT_STATUS set ACTIVE_UPGRADE_IND = 0;
BEGIN
    cmxzdt.v_zdt_ind:= -1;
END;
/
COMMIT;
```

36. Configure sequences for the passive environment to be even:

```
BEGIN
    CMXZDT.GET_ZDT_CONFIG;
    CMXZDT.CONFIGURE_SEQUENCES;
END;
/
```

37. Deploy and start the event queue in the passive environment:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
   CMXZDT.CONFIGURE_GGS_EVENT_REPLICAT(CMXZDT.zdt_local);
   CMXZDT.CONFIGURE_GGS_EVENT_EXTRACT(CMXZDT.zdt_local);
   CMXZDT.START_EVENT_QUEUE(out_error_message  => out_error_msg,
                            out_return_code => out_return_code);

   dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg =>
'||substr(out_error_msg,1,250));
```

```
END;
/
```

38. Deploy new replication baseline parameter files from the passive environment to the active environment:

```
BEGIN
    CMXZDT.CONFIGURE_GGS_EXTRACT(CMXZDT.zdt_local);
END;
/
```

39. Start extract and pump in the passive environment, which is now the source:

```
DECLARE
   out_error_msg VARCHAR2(32000);
   out_return_code INT;
BEGIN
  cmxzdt.start_replication_extract(in_env_type => cmxzdt.zdt_local,
                            out_error_message  => out_error_msg,
                               out_return_code => out_return_code);

  dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg => '||
substr(out_error_msg,1,250));
END;
/
```

40. Get the current SCN from the passive environment:

```
SQL-CMX_ORS_B> select current_scn from v$database;

CURRENT_SCN
-----------
    2880593
```

41. Export the passive environment using Data Pump with the SCN:

```
c:> <ors username>/<password>@<tns entry name>
    directory=<DATA_PUMP_DIR_OBJECT>
    dumpfile=<mrm_backup_envb.dmp>
    logfile=<mrm_backup_after_upgrade.log>
    parallel=8
    job_name=<EXPORT_AFTER_UPGRADE>
    flashback_scn=<CURRENT_SCN from the previous step>
```

42. Resume batch jobs in the passive environment.

Upgrade Steps Controlled from the Active Environment

The active environment must be prepared after you complete the upgrade steps on the passive environment. Run the steps from the active environment.

Important: During the upgrade, you must drop the source schema, re-create it from the target schema, and then import the database dump file. Do not attempt to bypass this process by applying a change list, because the schemas must be exactly the same in both databases for the replication to work. To avoid making inadvertent changes, enable Production Mode on the source and target databases. Log in to the Hub Console, select the Databases tool, select the database, and enable Production Mode. In future, if you need to apply a change list to the target database, you can disable Production Mode and apply the change list.

Before you begin, open the following repository tables and make a note of the values in the columns.

C_REPOS_ZDT_STATUS

Record the values for all columns. You need these values in step 6.

C_REPOS_DB_RELEASE

Record the values for the following columns. You need these values in step 7.

- db_password, tns_name
- connection_port
- oracle_sid
- database_host
- connect_url
- database_id
- connection_type
- proxy_ind
- db_proxy_username
- db_proxy_password
- db_replication_username
- db_replication_password
- debug_ind
- debug_level
- debug_file_name
- debug_file_pat

1. **Infrastructure Upgrade**. Upgrade hardware and third-party software in the active environment.
2. Stop the application servers, close connections, and then drop and re-create the schema in the active environment.
   a. Stop the MDM application servers at the active environment.
   b. Close all the connections, such as SQL*Plus, TOAD, and the application server that are connected to the active environment.
   c. Drop the schema in active environment, and then create it.

```
Drop schema A - using system user (sqlplus system/password@tnsname)
 SQL> drop user envA cascade;

Create schema A - using system user (sqlplus system/password@tnsname)
 SQL> <hub_server_install>/resources/database/custom_scripts/oracle/import/
@mk_cmx_ors_user; -- supply the schema name as A
```

3. Re-create the Operational Reference Store by importing the dump file that was generated from the passive environment.

```
C:\> impdp <dba_username>/<dba_password>@<tns_entry_name>
     directory=<DATA_PUMP_DIR_OBJECT>
     dumpfile==<mrm_backup_envb.dmp>
     logfile=<mrm_restore_after_upgrade.log>
     content=all
     remap_schema=<from_user>:<to_user>
     parallel=8
     job_name=<RESTORE_ENVB>
```

While the schema is being created, you might see the following messages:

```
ORA-39083: Object type TYPE failed to create with error:
ORA-02304: invalid object identifier literal
```

The type already exists, and therefore it is not re-created. You can safely ignore these messages.

4. Remove event queues and drop tables.
   a. Remove the ZDT event queue in the passive environment.

      Important: Do not drop the GGS_CHECKPOINT and GGS_CHECKPOINT_LOX tables. If you drop these tables, the REPLICAT service does not start.

      ```
      /* Repository tables for ZDT */
      delete from C_REPOS_ZDT_EVENT_QUEUE;
      delete from C_REPOS_ZDT_REPLICAT_EXCEPTION;
      update C_REPOS_ZDT_ENV_STATE set state = NULL, state_ts = NULL, state_desc =
      NULL, updated_by=NULL, update_date=NULL;

      /* Tables for Oracle GoldenGate */
      delete from GGS_CHECKPOINT;        -- Do not drop this table
      delete from GGS_CHECKPOINT_LOX;    -- Do not drop this table

      drop table GGS_EVENT_CHECKPOINT cascade constraints;
      drop table GGS_EVENT_CHECKPOINT_LOX cascade  constraints;
      drop table GGS_EVENT_TRACE cascade  constraints;
      commit;
      ```

   b. Search for %GGS% tables. Verify that the GGS_CHECKPOINT and GGS_CHECKPOINT_LOX tables exist. If you find any other tables with GGS in the table name, including TMP_GGS_* tables, you can drop these other tables.
5. Re-compile the packages, stored procedures, and views.
   a. Navigate to `<MDM installation directory>/resources/database/oracle/en_US/`.
   b. As the SYSTEM user, run the following commands from SQL*plus:

      ```
      SQL > @update_javasp.sql
            @compile_types.sql
      ```

   c. Navigate to `<MDM installation directory>/resources/database/oracle/`.
   d. Run the following commands from SQL*plus:

      ```
      SQL > @cmx_zdt_objects_support.sql
            @cmxlb_pack.plb
            @cmx_debug_print_prc.plb
            @cmxlog_pack.plb
            @cmxlog_body.plb
            @cmxzdt_pack.plb
            @cmxzdt_body.plb
      ```

6. Restore the ZDT status for the active environment.

   Use the values that were in the C_REPOS_ZDT_STATUS table before you dropped the schema.

   ```
   delete from c_repos_zdt_status;
   commit;

   insert into C_REPOS_ZDT_STATUS
   (REPLICATION_TARGET_IND, ACTIVE_UPGRADE_IND, CREATOR, CREATE_DATE,GGS_HOME_PATH,
   REPLICAT_NUMBER, TRAIL_FILE_SIZE, DISCARD_FILE_SIZE, ACCEPTABLE_LAG_MINUTES,
   LAG_DETECTION_TOKEN, LOCAL_ENVIRONMENT_NAME, LOCAL_SCHEMA_NAME, LOCAL_TRAIL_PATH,
   PUMP_RMTHOST, PUMP_MGRPORT, REMOTE_TRAIL_PATH, REMOTE_ENVIRONMENT_NAME,
   REMOTE_SCHEMA_NAME , REGULAR_STREAM_ID, EVENT_QUEUE_ID, EXTRACT_PREFIX,
   REPLICAT_PREFIX , BATCH_DISABLED_IND, DEFAULT_TIMEOUT_MINUTES
   )
   Values
   (<insert a comma-separated list of the values that were in the C_REPOS_ZDT_STATUS
   table before you dropped the schema.>
   );
   commit;
   ```

7. Check the environment-specific settings on the active environment in the C_REPOS_DB_RELEASE table.

All the values in the table must be local and the database entries must point to the local database. If necessary, update the values to match the values that were in the C_REPOS_DB_RELEASE table before you dropped the schema.

8. Configure sequences on the active environment to be odd:

```
BEGIN
    CMXZDT.GET_ZDT_CONFIG;
    CMXZDT.CONFIGURE_SEQUENCES;
END;
/
```

9. Deploy and start event queues on the active environment, which is the new target:

```
DECLARE
out_error_msg VARCHAR2(32000);
out_return_code INT;
BEGIN
  CMXZDT.CONFIGURE_GGS_EVENT_REPLICAT(CMXZDT.zdt_local);
  CMXZDT.CONFIGURE_GGS_EVENT_EXTRACT(CMXZDT.zdt_local);
  CMXZDT.START_EVENT_QUEUE(out_error_message  => out_error_msg,
                out_return_code => out_return_code);
  dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg => '||
substr(out_error_msg,1,250));
END;
/
```

10. Deploy new replication baseline parameter files from the active environment to the passive environment:

```
BEGIN
  CMXZDT.CONFIGURE_GGS_REPLICAT(CMXZDT.zdt_local);
END;
/
```

11. Start replication replay on the active environment. Use the SCN you obtained in "Upgrade Steps Controlled from the Passive Environment" on page 10.

```
DECLARE
out_error_msg VARCHAR2(32000);
out_return_code INT;
begin
  cmxzdt.start_replication_replay(in_env_type => cmxzdt.zdt_local,
                                  out_error_message => out_error_msg,
                                  out_return_code => out_return_code,in_after_csn
=> <SCN>);

  dbms_output.put_line('out_return_code => '||out_return_code||' out_error_msg => '||
substr(out_error_msg,1,250));
end;
/
```

12. **MDM Upgrade**. On the new passive environment, upgrade the Multidomain MDM software and configure the process server.

CHAPTER 3

Troubleshooting

This chapter includes the following topics:

Batch job fails when backfill tasks are registered

If you run batch jobs with backfill tasks enabled, an error occurs and the batch job fails.

The MDM Hub does not allow batch jobs to run in a target replication environment while backfill tasks are registered. The MDM Hub does not allow batch jobs to run because of the high likelihood of locking conflicts when backfill batch jobs are running concurrently with regular batch jobs. After the backfill task has completed and the backfill tasks are de-registered, you can run batch jobs.

Reset Matches on Target

The Reset Match logic is not replicated to the target environment. MET migration will not reset the match table. You must do this using a script. Tokenize backfill does not impact the match table. After a match-related change list is applied, a change must be made to the target match rules in order to trigger Reset Match.

Message Type Enhancement

The message type enhancement can set up GoldenGate and switch ZDT schema processes. The ZDT schema process can switch from either the source environment or the target environment, though switching from the target environment is preferred.

Use the message type enhancement for the following use cases:

Update remote tables.

Run the following command to update remote tables:

```
Target C_REPOS_ZDT_STATUS.ACTIVE_UPGRADE_IND will set to 1 from source.
begin
cmxzdt.send_event_message(in_type=>cmxzdt.zdt_execute, in_parameters=>'update
c_repos_zdt_status set active_upgrade_ind=1' );
end;
/
```

Call remote stored procedures.

Run the following command to generate extract processes and PRM files from the target environment:

```
begin
cmxzdt.send_event_message( in_type=>cmxzdt.zdt_execute, in_parameters=>'BEGIN
CMXZDT.CONFIGURE_GGS_EXTRACT(CMXZDT.zdt_local); END;' );
end;
/
```

The message type enhancement cannot run any logic that uses messages recursively.

Message queue replication is not working

If the cmxzdt.configure_ggs_replicate script does not complete, the ZDT message queue replication might not work between the source database and the target database.

1. Check that all the Oracle GoldenGate processes are running. Restart any processes that are not in the RUNNING state.
   In this example, ENVA contains the source database and ENVB contains the target database.

   ```
   EXTRACT RUNNING EQENVA
   EXTRACT RUNNING E_ENVA
   EXTRACT RUNNING PQENVA
   EXTRACT RUNNING P_ENVA
   REPLICAT RUNNING RQENVA

   EXTRACT RUNNING EQENVB
   EXTRACT RUNNING PQENVB
   REPLICAT RUNNING RQENVB
   REPLICAT ABENDED R_ENVB
   REPLICAT RUNNING R_ENVBU
   ```

   In this example, the R_ENVB process is in the ABENDED state. Try restarting the process.
2. Verify that the message queue replication is working in both directions.
   Insert an event directly into the C_REPOS_ZDT_EVENT_QUEUE table in the source database. Open the same table in the target database. If the event appears in the target database table, replication is working in this direction. Repeat the verification process from the target database to ensure that the replication works in the other direction as well.

   For example, the following code adds an event to the table on ENVA:

   ```
   insert into C_REPOS_ZDT_EVENT_QUEUE ( 'enva', -1, 'test', '', 'envb', 'test',
   CURRENT_TIMESTAMP, 'EVENT_TOKEN' );
   ```

   The following code adds an event to the table on ENVB:

   ```
   insert into C_REPOS_ZDT_EVENT_QUEUE ( 'envb', -1, 'test', '', 'enva', 'test',
   CURRENT_TIMESTAMP, 'EVENT_TOKEN' );
   ```
3.

4. If the Oracle GoldenGate processes are running without errors, but the message queue replication is not working, you need to troubleshoot your environment. Navigate to the Oracle GoldenGate directory `dirrpt` and check the .rpt files for information about potential problems.
   For more information about replication issues, see the following Oracle articles on Metalink:
   1. Main Note - Oracle GoldenGate - Troubleshooting (Doc ID 1306476.1)
   2. Master Note - Oracle GoldenGate: Initial Load Techniques and References (Doc ID 1311707.1)
   3. DB Transactions Missing from Oracle GoldenGate Trail Files (Doc ID 1364852.1)
   4. POC for golden gate

INDEX