Implementing a Persistent Identifier Module in MDM Multidomain Edition
Abstract

In an Informatica MDM Hub implementation, you might want to persist a unique identifier on the record in the MDM Hub that is the best version of the truth. The unique identifier is often on the primary parent base object table, such as the PARTY table for persons and organizations. Due to the complex nature of the MDM Hub, where data records are matched and merged across multiple data sources, the unique identifier changes after records are updated. Typically the most recent identifier value of a cross-reference (XREF) record survives.

The core idea of a persistent unique identifier for the base object record is to determine which XREF record that holds the unique identifier associated with the base object record survives.

The Persistent Identifier Module works with Oracle and Microsoft SQL Server databases. To get the Persistent Identifier Module Java JAR file, contact Informatica Global Customer Support.

Supported Versions

- MDM Multidomain Edition 9.7.1
- MDM Multidomain Edition 10.0.0
- MDM Multidomain Edition 10.1.0

Table of Contents

Overview ................................................................. 3
Survivorship Types ....................................................... 3
Identifier Types .......................................................... 3
Unique Identifiers .......................................................... 4
Persistent Identifier Module Implementation ............................. 4
  Step 1: Enable Base Object Table History ............................ 5
  Step 2: Enable Trust and Validate Option on Base Object Column ........................................ 5
  Step 3: Adjust Trust Setting on the Persistent ID Column ........................................ 5
  Step 4. Setting Custom Validation Rule on the Persistent ID Column ........................................ 6
C_REPOS_PERSIST_COLUMN_CONFIG Table Definition ............... 7
  Additional Configuration of C_REPOS_PERSIST_COLUMN_CONFIG ........................................ 10
C_LOG_RESET_DLT_TO_ACTV Table Definition ......................... 10
Persistent ID Module User Exits ........................................ 12
  Post-Load ............................................................... 12
  Post-Merge ............................................................. 13
  Pre-Unmerge .......................................................... 13
  Post-Unmerge ........................................................ 14
  Post-Stage ............................................................. 14
  Bypassing Stage Processing with a Post-Stage User Exit ........................................ 15
MDM Persist ID User Exits Implementation ................................ 16
  Step 1. Identify or Create the Column to Hold the Persistent ID ........................................ 16
  Step 2. Create the Configuration and Log Tables ........................................ 16
  Step 3. Register the Unique ID Column ........................................ 17
Overview

The MDM Persistent Identifier Module is a Java library, or package, that MDM Hub user exits can invoke. The Persistent Identifier Module works with Oracle and Microsoft SQL Server databases.

The module provides configurable survivorship rules to determine which persistent ID for a cross-reference record survives on the base object record. The survivorship, or persisting, of the identifier value acts as a custom best version of the truth (BVT) calculation. The BVT calculation uses the MDM Hub trust framework to downgrade losing cross-reference records, and does not downgrade the winning cross-reference record for the associated unique identifier column.

The following MDM Hub user exits work with the Persistent Identifier Module:

- Post-Load
- Post-Merge
- Pre-Unmerge
- Post-Unmerge
- Post-Stage

**Restriction:** You can only use the Persistent Identifier Module with a Post-Stage user exit under specific conditions.

To get the Persistent Identifier Module Java JAR file, contact Informatica Global Customer Support. The file is named MDMPersistentId_<version#>_jar.

Survivorship Types

The Persistent Identifier Module introduces custom trust calculations that affect survivorship on base object records. The following types of survivorship rules on the unique identifier column are incorporated within the Persistent Identifier Module:

- The oldest cross-reference record identifier value in the merged base object records survives on the base object record.
- The cross-reference record identifier value in the merged base object records where the rowid_object equals the orig_rowid_object survives on the base object record.

Identifier Types

The following types of identifiers work with the Persistent Identifier Module:

**Identifier loaded from source systems**

If identifier uniqueness is required, the source system must provide and guarantee the unique identifier value in the MDM Hub landing tables.
Identifier is the MDM rowid_object value

The Persistent Identifier Module considers the rowid_object value as a MDM-generated identifier. The rowid_object is copied into the user column that is identified as the unique ID column.

Identifier is the MDM rowid_xref value

The Persistent Identifier Module considers the rowid_xref value as a MDM-generated identifier. The rowid_xref value is copied into the user column that is identified as the unique ID column.

Identifier generated by ETL process or by the MDM Post-Landing processes

The Persistent Identifier Module considers the generated identifier as though it was coming from source systems. The identifier must be in the MDM Hub landing table to load into base object and XREF tables.

Unique Identifiers

The unique identifier column is a standard MDM user-defined column that contains a value on a base object table that has trust and validation enabled. For the Persistent Identifier Module to function, every source record created in the XREF table of the base object must have a unique identifier. The source system that supplies the identifier must guarantee the uniqueness of the identifier.

Identifiers can also be generated by an ETL process or generated by a post-landing process. The Persistent Identifier Module considers these identifiers as if they came from source systems.

The unique identifier generated by the MDM Hub are either based on the rowid_object or the rowid_xref value when records are inserted into MDM. The rowid_object or the rowid_xref values can also be formatted through a database format function.

Only one identifier column for each base object table is supported in the MDM Hub. The Persistent Identifier Module works with user exits that can prolong MDM processes. Informatica recommends that you configure a maximum of three base object tables with unique identifiers for persistent ID processing within an MDM Operational Reference Store (ORS).

Persistent Identifier Module Implementation

The Persistent Identifier Module is a Java library that MDM Hub user exits can invoke. The module provides configurable survivorship rules to determine which persistent ID for a cross-reference record survives on the base object record. The Persistent Identifier Module is compatible with the following user exits:

- Post-Load
- Post-Merge
- Pre-Unmerge
- Post-Unmerge
- Post-Stage

Restriction: You can use the Persistent Identifier Module with a Post-Stage user exit under specific conditions. At the end of a batch stage process, you can use the Persistent Identifier Module to determine if the unique identifier value for an existing cross-reference record will change. For those identifier values that will change, the Persistent Identifier Module updates the stage record unique identifier column with the corresponding identifier value of the cross-reference record. This prevents the identifier value of the cross-reference record from changing while the stage records are loaded into the XREF table.

To enable the Persistent Identifier Module, you must perform these steps:
Step 1: Enable Base Object Table History

You must enable history on the base object table that requires the persistent ID process. The Persistent Identifier Module uses the history data to reset the unique identifier value if it was changed.

1. Acquire a write lock.
2. In the Model workbench, click Schema.
3. Select the base object with the Persistent ID column.
4. In Base Object Properties, select the Enable History check box.

Step 2: Enable Trust and Validate Option on Base Object Column

You must enable the Trust and Validate options on the user column on the base object table that contains the record unique identifier. The Persistent Identifier Module uses the MDM trust framework to persist the unique identifier column value.

Note: In Informatica Data Director (IDD), you must set the unique identifier column to be non-editable if the MDM Hub generates the unique identifier. You do this through the IDD Configuration Manager. IDD users must not be able to edit the unique identifier value.

1. Acquire a write lock.
2. In the Model workbench, click Schema.
3. Select the column on the base object that contains the record unique identifier.
4. Select the Trust and Validate check boxes.

Step 3: Adjust Trust Setting on the Persistent ID Column

Set the minimum and maximum trust to the same values across all source systems on the unique identifier column. Informatica recommends that you set the minimum trust and maximum trust values to 80 across all source systems.

1. Acquire a write lock.

Step 4: Set Custom Validation Rule on the Persistent ID Column

To preserve the Persistent ID BVT calculation during the MDM revalidate process, you must set a custom validation rule on the Persistent ID column.

Note: Depending on the MDM Hub version, the Custom Validation option might not be available in an Microsoft SQL Server environment.

You must enable history on the base object table that requires the persistent ID process. Enable base object table history in the Schema tool of the Hub Console.

1. Acquire a write lock.
2. In the Model workbench, click Schema.
3. Select the base object with the Persistent ID column.
4. In Base Object Properties, select the Enable History check box.

Step 2: Enable Trust and Validate Option on Base Object Column

The Persistent Identifier Module uses the MDM trust framework to persist the unique identifier column value. Enable trust and validation in the Schema tool of the Hub Console.

1. Acquire a write lock.
2. In the Model workbench, click Schema.
3. Select the column on the base object that contains the record unique identifier.
4. Select the Trust and Validate check boxes.

Step 3: Adjust Trust Setting on the Persistent ID Column

Set the minimum and maximum trust to the same values across all source systems on the unique identifier column. Informatica recommends that you set the minimum trust and maximum trust values to 80 across all source systems. Adjust trust settings in the Systems and Trust tool of the Hub Console.

1. Acquire a write lock.
2. In the Model workbench, click **Systems and Trust**.

3. In the list of source systems, select the source system that you want to configure.
   The screen refreshes, showing the **Edit** button next to the name and description fields for the selected source system.

4. Change the maximum and minimum trust settings to 80 for each source system that has the Persistent ID column.

**Step 4. Setting Custom Validation Rule on the Persistent ID Column**

The Persistent ID Module requires a custom validation rule on the Persistent ID column. Set validation rules in the Schema tool of the Hub Console.

1. Acquire a write lock.

2. In the Model workbench, click **Schema**.

3. Select **Validation Rules Setup** for the base object with the Persistent ID column.
   The Validation Rules page appears.

4. Click the **Add validation rule** button.
   The Add Validation Rule dialog box appears.

5. Enter a rule name and select the **Custom** rule type from the list.

6. For downgrade percentage, select **90** and clear the **Retain Minimum Trust** check box.

7. In the Rule SQL section, enter the following code:

   ```sql
   WHERE EXISTS
   (SELECT 1
    FROM <bo_table_name> XREF X
    , <bo_table_name> XREF V
    WHERE I.PKEY_SRC_OBJECT = X.PKEY_SRC_OBJECT
    AND I.ROWID_SYSTEM = X.ROWID_SYSTEM
    AND I.HUB_STATE_IND = 1
    AND V.ROWID_XREF = X.ROWID_XREF
    AND V.<id_column_name>_VOR = 0
   )
   ```

   The following image shows an example of the custom validation rule configured in the Hub Console:
C_REPOS_PERSIST_COLUMN_CONFIG Table Definition

To support the Persistent Identifier Module with configurable options, the custom MDM repository configuration table C_REPOS_PERSIST_COLUMN_CONFIG contains the unique identifier column configuration for persistent ID processing. The Persistent Identifier Module can use only one column for each base object.

The following table describes the definition of the C_REPOS_PERSIST_COLUMN_CONFIG table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Constraint and Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWID_PERSIST_COLUMN</td>
<td>CHAR(14)</td>
<td>Not Null.</td>
<td>Contains a ROWID value that is consistent with other MDM REPOS tables.</td>
</tr>
<tr>
<td>ROWID_COLUMN</td>
<td>CHAR(14)</td>
<td>Not Null. Unique Value</td>
<td>Contains the ROWID_COLUMN value from the C_REPOS_COLUMN table to identify the unique ID for persistent ID processing.</td>
</tr>
<tr>
<td>ENABLED_IND</td>
<td>INTEGER</td>
<td>Not Null. Default is 1</td>
<td>Set to 0 to turn off Persistent ID processing on the base object table column. Set to 1 to turn on Persistent ID processing on the base object table column.</td>
</tr>
<tr>
<td>Field</td>
<td>Data Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SURVIVORSHIP_TYPE</td>
<td>INTEGER</td>
<td>Not Null. Value is 0 or 1. Set to 0 for oldest XREF survivorship, by create date and lowest ROWID_XREF. Set to 1 for Target ROWID_OBJECT. The winning Persistent ID value is the XREF record where the ROWID_OBJECT equals ORIG_ROWID_OBJECT. Note: For value of 1, both IGNORE_XREF_HSI and IGNORE_XREF_TIMELINE are not used, and both are ignored.</td>
<td></td>
</tr>
<tr>
<td>UNIQUE_ID_TYPE</td>
<td>INTEGER</td>
<td>Not Null. Value can be 0 to 3. Set to 0 if the ID value is from source systems, or was loaded from landing tables, through the stage process. Set to 1 to use the ORIG_ROWID_OBJECT as the unique ID value. Set to 2 to use the ROWID_XREF as the unique ID value. Set to 3 to use UNIQUE_ID_FORMAT_FUNCTION as the unique ID value.</td>
<td></td>
</tr>
<tr>
<td>UNIQUE_ID_FORMAT_FUNCTION</td>
<td>VARCHAR2(255)</td>
<td>Nullable. Must be included if UNIQUE_ID_TYPE is set to 3. A string with the format function for a unique ID creation, or a simple function to return a unique ID. For example, format the ROWID_XREF to 12 digits and prefix it with PA. Oracle functions return: 'PA'</td>
<td></td>
</tr>
<tr>
<td>ALLOW_UNMERGE_XREF_WITH_ID</td>
<td>INTEGER</td>
<td>Not Null. Default is 0. Set to 0 to not allow XREF record holding the Persistent ID to be unmerged in real-time and in Batch Unmerge. Set to 1 to allow the Unmerge.</td>
<td></td>
</tr>
<tr>
<td>ALLOW_DELETE_XREF_WITH_ID</td>
<td>INTEGER</td>
<td>Not Null. Default is 0. Set to 0 to not allow XREF record holding the Persistent ID value to set HUB_STATE_IND to -1. Set to 1 to allow XREF record holding the Persistent ID value to set HUB_STATE_IND to -1. Note: In real-time, the Delete SIF API does not enter user exits and therefore the delete cannot be rejected/aborted by throwing an error. In batch, during the stage process, it will reactivate the stage record that would hub state delete the XREF record holding the Persistent ID. The reactivation is logged in table C_LOG_RESET_DEL_TO_ACTV for the ROWID_JOB.</td>
<td></td>
</tr>
<tr>
<td>COLUMN NAME</td>
<td>DATA TYPE</td>
<td>NOT NULL</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>IGNORE_XREF_HSI</td>
<td>INTEGER</td>
<td>Not Null</td>
<td>Default is 0.</td>
</tr>
<tr>
<td>IGNORE_XREF_TIMELINE</td>
<td>INTEGER</td>
<td>Not Null</td>
<td>Default is 0.</td>
</tr>
<tr>
<td>DOWNGRADE_LOSING_XREF_PERCENT</td>
<td>INTEGER</td>
<td>Not Null</td>
<td>Value between 0 and 100. Default is 90.</td>
</tr>
<tr>
<td>JOB_METRIC_TYPE_CODE</td>
<td>INTEGER</td>
<td>Not Null</td>
<td>Default is 101.</td>
</tr>
<tr>
<td>UNMERGE_REJECT_MSG</td>
<td>VARCHAR2(1024)</td>
<td>Not Null</td>
<td></td>
</tr>
<tr>
<td>MERGE_LIST_EVAL_SIZE</td>
<td>INTEGER</td>
<td>Not Null</td>
<td>Default is 300.</td>
</tr>
<tr>
<td>PRECEDENCE_SOURCE_SYSTEMS</td>
<td>VARCHAR2(255)</td>
<td>Nullable</td>
<td></td>
</tr>
</tbody>
</table>
### Additional Configuration of C_REPOS_PERSIST_COLUMN_CONFIG

The following table describes additional configuration notes on the columns defined in the C_REPOS_PERSIST_COLUMN_CONFIG table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIQUE_ID_FORMAT_FUNCTION</td>
<td>To get a calculated record identifier, you can configure the UNIQUE_ID_FORMAT_FUNCTION as a stored procedure function that the MDM Persistent ID invokes. If you do configure the function as a stored procedure, it must be efficient enough to handle the MDM load process insert cycle.</td>
</tr>
</tbody>
</table>
| UNMERGE_REJECT_MSG               | The UNMERGE_REJECT_MSG column contains an implementation custom unmerge error message that the MDM Pre-Unmerge user exit uses when ALLOW_UNMERGE_XREF_WITH_ID is set to 0. The following placeholder columns can be embedded into the message string:  
  - $(ROWID_OBJECT) - The XREF record rowid_object value.  
  - $(ROWID_XREF) - The XREF record rowid_xref value.  
  - $(PKEY_SRC_OBJECT) - The XREF record PKEY_SRC_OBJECT value.  
  - $(ROWID_SYSTEM) - The XREF record rowid_system value.  
  - $(HUB_STATE_IND) - The XREF record hub state value.  
  - $(PID_COLUMN_VALUE) - The XREF record ID column value.  
  - $(RETURN_CODE) - The return code of the unmerge (-101).  
  Note: Enter column names in uppercase.  
  For example, you use the following custom unmerge error message set in the UNMERGE_REJECT_MSG column:  
  The cross-reference record being unmerged with PKEY_SRC_OBJECT "$PKEY_SRC_OBJECT" and ROWID_SYSTEM "$ROWID_SYSTEM" holds the Persistent ID value for base object record with ROWID_OBJECT of "$ROWID_OBJECT", and cannot be unmerged. Return code "$RETURN_CODE".  
  The example message in Informatica Data Director, or generated in real-time PUT or cleansePUT, appears as follows:  
  The cross-reference record being unmerged with PKEY_SRC_OBJECT "X010224" and ROWID_SYSTEM "VVA" holds the Persistent ID value for base object record with ROWID_OBJECT of "10034", and cannot be unmerged. Return code -101. |
| ALLOW_DELETE_XREF_WITH_ID        | In real time, HUB_STATE_IND deletes through the SIF Delete API to set the XREF record of HUB_STATE_IND to -1. The MDM Hub does not invoke any user exits to intercept the Hub delete for additional processing. Before using the Hub delete, the composite service can call a custom graphics function to perform a read to determine if the XREF can be soft deleted. |

### C_LOG_RESET_DLT_TO_ACTV Table Definition

The Persistent ID process uses the C_LOG_RESET_DLT_TO_ACTV table when the ALLOW_DELETE_XREF_WITH_ID table is disabled during the post-stage MDM Persistent ID processing. Some stage table records have soft-deleted XREF records holding the Persistent ID value and XREF records that have more than
one active merge records. The Persistent ID process reactivates these stage table records by resetting the Hub Stage Ind to 1. The C_LOG_RESET_DLT_TO_ACTV table contains the log of the resets.

The following table describes the definition of the C_LOG_RESET_DLT_TO_ACTV table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Constraint and Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_DATE</td>
<td>TIMESTAMP</td>
<td>Nullable</td>
<td>The creation date of the log record.</td>
</tr>
<tr>
<td>ROWID_JOB_CNTRL</td>
<td>CHAR(14)</td>
<td>Nullable</td>
<td>The MDM ROWID_JOB (job ID) associated with the C_REPOS_JOB_CONTROL table job entry.</td>
</tr>
<tr>
<td>STG_TABLE_NAME</td>
<td>VARCHAR2(50)</td>
<td>Nullable</td>
<td>The stage table name.</td>
</tr>
<tr>
<td>XREF_TABLE_NAME</td>
<td>VARCHAR2(50)</td>
<td>Nullable</td>
<td>The cross-reference table name associated with the stage table.</td>
</tr>
<tr>
<td>STG_SRC_ROWID</td>
<td>VARCHAR2(30)</td>
<td>Nullable</td>
<td>The SRC_ROWID value of the stage record.</td>
</tr>
<tr>
<td>STG_LAST_UPDATE_DATE</td>
<td>TIMESTAMP</td>
<td>Nullable</td>
<td>The date that the stage record was last updated.</td>
</tr>
<tr>
<td>STG_PKEY_SRC_OBJECT</td>
<td>VARCHAR2(255)</td>
<td>Nullable</td>
<td>The PKEY_SRC_OBJECT value of the stage record.</td>
</tr>
<tr>
<td>STG_HUB_STATE_IND</td>
<td>INTEGER</td>
<td>Nullable</td>
<td>The HUB_STATE_IND value of the stage record.</td>
</tr>
<tr>
<td>STG_VERSION_SEQ</td>
<td>INTEGER</td>
<td>Nullable</td>
<td>The version sequence value of the stage record.</td>
</tr>
<tr>
<td>STG_PERIOD_START_DATE</td>
<td>TIMESTAMP</td>
<td>Nullable</td>
<td>The period start date of the stage record.</td>
</tr>
<tr>
<td>STG_PERIOD_END_DATE</td>
<td>TIMESTAMP</td>
<td>Nullable</td>
<td>The period end date of the stage record.</td>
</tr>
<tr>
<td>STG_PERSIST_ID_VALUE</td>
<td>VARCHAR2 (255)</td>
<td>Nullable</td>
<td>The Persist ID (unique ID) value of the stage record.</td>
</tr>
<tr>
<td>XREF_ROWID_XREF</td>
<td>INTEGER</td>
<td>Nullable</td>
<td>The corresponding XREF record ROWID_XREF value of the stage record.</td>
</tr>
<tr>
<td>XREF_ROWID_OBJECT</td>
<td>CHAR(14)</td>
<td>Nullable</td>
<td>The corresponding XREF record ROWID_OBJECT value of the stage record.</td>
</tr>
<tr>
<td>XREF_PKEY_SRC_OBJECT</td>
<td>VARCHAR(255)</td>
<td>Nullable</td>
<td>The corresponding XREF PKEY_SRC_OBJECT value of the stage record.</td>
</tr>
<tr>
<td>XREF_ROWID_SYSTEM</td>
<td>CHAR(14)</td>
<td>Nullable</td>
<td>The corresponding XREF ROWID_SYSTEM value of the stage record.</td>
</tr>
<tr>
<td>XREF_HUB_STATE_IND</td>
<td>INTEGER</td>
<td>Nullable</td>
<td>The corresponding XREF HUB_STATE_IND value of the stage record.</td>
</tr>
<tr>
<td>XREF_PERIOD_START_DATE</td>
<td>TIMESTAMP</td>
<td>Nullable</td>
<td>The corresponding XREF period start date of the stage record.</td>
</tr>
</tbody>
</table>
The corresponding XREF period end date of the stage record.

| XREF_PERSIST_ID_VALUE | VARCHAR2 (255) | Nullable | The corresponding XREF Persist ID (unique ID) value of the stage record. |

**Persistent ID Module User Exits**

The Persistent ID Module is a Java library called com.informatica.mdm.ue.persistid. Within the library, also called a package, there are two public classes: PersistId and PersistIdExceptions.

**PersistIdException**

The error exception class.

**PersistId**

The main class that is instantiated with constructor parameters to identify the user exit type and the MDM user exit parameters. The PersistId class then invokes the main method executePersistId, which starts the Persistent ID process.

To invoke the Persistent ID process, the MDM user exits require the following import statements:

- import com.informatica.mdm.ue.persistid.PersistId
- import com.informatica.mdm.ue.persistid.PersistIdExceptions

The executePersistId can throw either PersistIdExceptions or standard Java exceptions. When a user exit calls the Persistent ID process, it instantiates the PersistId class and runs the method executePersistId to start the Persistent ID process.

For more information about user exits, see the Informatica MDM Multidomain Edition Configuration Guide.

**Post-Load**

The MDM Persistent ID Module performs two functions in the post-load user exit.

For record inserts, the Persistent ID Module considers the UNIQUE_ID_TYPE column in the C_REPOS_PERSIST_COLUMN configuration table. If the UNIQUE_ID_TYPE column is set to 1, 2, or 3, the Persistent ID Module assigns values to the unique ID column according to the UNIQUE_ID_TYPE column setting. If the UNIQUE_ID_TYPE column is set to 0, the Persistent ID Module does not process the inserted records and accepts the unique ID column as it is.

On the update cycle of the load process, the Persistent ID Module determines the surviving value on the base object record as defined in the SURVIVORSHIP_TYPE of the C_REPOS_PERSIST_COLUMN configuration table. It also considers the IGNORE_XREF_HSI and IGNORE_XREF_TIMELINE column settings to determine the XREF unique ID survivorship.

In real-time updates, before it determines the XREF unique ID survivorship, the Persistent ID Module compares the inbound unique ID value against the old unique ID value of the cross-reference record. If the inbound unique ID value is different, the Persistent ID Module resets the value to the prior unique ID value from the cross-reference record history. In batch processing, the user exit in the stage process handles the reset of the unique ID value.

**Example of Post-Load User Exit Invoking Persistent ID Module**

The following code is an example of a post-load user exit invoking the PersistId class:

```java
PersistId persistId = new PersistId( PersistId.UserExitType.POST_LOAD, userExitContext, actionType
```
Post-Merge

In the post-merge user exit, the Persistent ID Module determines the surviving value as defined in the SURVIVORSHIP_TYPE of the C_REPOS_PERSIST_COLUMN configuration table.

The Persistent ID Module also considers the IGNORE_XREF_HSI and IGNORE_XREF_TIMELINE options to determine the XREF unique identifier value that wins on the base object. It then downgrades the data lineage on the unique ID column for the losing XREF records.

Example of Post-Merge User Exit Invoking Persistent ID Module

The following code is an example of a post-merge user exit invoking the PersistId class:

```java
PersistId persistId = new PersistId( PersistId.UserExitType.POST_MERGE,
                                    userExitContext, rowidObjectMapList );
try {
    persistId.executePersistId();
} catch (PersistIdExceptions pidEx) {
    throw new Exception(pidEx.getMessage());
} catch (Exception ex) {
    throw new Exception(ex);
}
```

Pre-Unmerge

In the pre-unmerge user exit, the Persistent ID Module determines if the XREF that contains the persistent ID can be unmerged. Permission for an unmerge is defined in ALLOW_UNMERGE_XREF_WITH_ID column of the C_REPOS_PERSIST_COLUMN configuration table.

If the unmerge is not permitted, the MDM Hub aborts the unmerge and generates the error message configured in UNMERGE_ERROR_MSG. If the unmerge is permitted, the unmerge process continues.

Example of Pre-Unmerge User Exit Invoking Persistent ID Module

The following code is an example of a pre-unmerge user exit invoking the PersistId class:

```java
PersistId persistId = new PersistId( PersistId.UserExitType.PRE_UNMERGE,
                                    userExitContext,
                                    postUnmergeResponse.size() );
try {
    persistId.executePersistId();
} catch (PersistIdExceptions pidEx) {
    throw new Exception(pidEx.getMessage());
} catch (Exception ex) {
    throw new Exception(ex);
}
```
**Post-Unmerge**

In the post-unmerge user exit, the Persistent ID Module performs a BVT calculation on the unique ID column twice. It runs the calculation on the source rowid_object of the unmerged base object, and then runs the calculation on the target rowid_object where the un-merge occurred.

On each rowid_object, the Persistent ID Module performs the unique ID survivorship based on the definition in the SURVIVORSHIP_TYPE of the C_REPOS_PERSIST_COLUMN configuration table.

**Example of Post-Unmerge Invoking Persistent ID Module**

The following code is an example of a post-unmerge user exit invoking the PersistID class:

```java
PersistId persistId = new PersistId(PersistId.UserExitType.POST_UNMERGE
 , userExitContext, unmergeKey);
try {
    persistId.executePersistId();
} catch (PersistIdExceptions pidEx) {
    throw new Exception(pidEx.getMessage());
} catch (Exception ex) {
    throw new Exception(ex);
}
```

**Post-Stage**

Data sources that are both data consumers and data contributors for the MDM Hub can send incorrect or null unique identifiers in the data. After staging, if the MDM Hub accepts unfiltered staged records and updates the cross-reference records, the inbound data might corrupt the unique identifiers in the cross-reference records. This can also cause duplicate identifiers.

In a post-stage user exit, the Persistent ID Module compares the unique ID values on the stage table records against the corresponding cross-reference records to be updated. The Persistent ID Module examines PKEY_SRC_OBJECT and ROWID_SYSTEM. If the ID values of the stage table records are unique, the Persistent ID Module resets the value to the prior unique ID value from the cross-reference record history.

**Example of Post-Stage User Exit Invoking Persistent ID Module**

The following code is an example of a post-stage user exit invoking the PersistID class:

```java
PersistId persistId = new PersistId(PersistId.UserExitType.POST_STAGE
 , userExitContext, stageTableName, landingTableName
 , previousLandingTableName);
try {
    persistId.executePersistId();
} catch (PersistIdExceptions pidEx) {
    throw new Exception(pidEx.getMessage());
} catch (Exception ex) {
    throw new Exception(ex);
}
```
Bypassing Stage Processing with a Post-Stage User Exit

On Oracle databases, you might bypass the MDM staging process with an ETL process that inserts records directly into the stage table. However, this can cause duplicates and corrupt the unique identifiers in the cross-reference records.

To prevent the occurrence of duplicates or data corruption, the ETL process must invoke the MDM_STG_PERSISTID_RETENTION stored procedure package. You must first ensure the server or workstation where the Oracle client is installed meets the following prerequisites:

- SQL*Plus is installed
- The ORS schema where Persistent ID Module is deployed has the appropriate TNS configuration
- The Oracle client bin directory is in the PATH environment

Deploying MDM_STG_PERSISTID_RETENTION Stored Procedure Package

To deploy the MDM_STG_PERSISTID_RETENTION stored procedure package, perform the following steps:

1. Save the file `mdm_stgPersistidRetention.plb` into a directory on the server or workstation where the Oracle client is installed.
2. In SQL*Plus, change the current working directory to where you saved the `mdm_stgPersistidRetention.plb` file.
3. To login to the ORS schema, type the following command in SQL*Plus:
   ```sql
   sqlplus <ors_schema_user>@<tns_name>/.<password>
   ```
4. Compile and deploy the MDM_STG_PERSISTID_RETENTION package.
5. At the SQL prompt, enter `@ mdm_stgPersistidRetention.plb`;
   The package and package body are created.

MDM_STG_PERSISTID_RETENTION Stored Procedure Interface and Definition

The following code shows the interface to the MDM_STG_PERSISTID_RETENTION procedure:

```sql
MDM_STG_PERSISTID_RETENTION.EXEC_STG_PERSISTID_DIRECT
    ( in_stage_table_name  in VARCHAR2
    , out_err_msg          out VARCHAR2
    , out_err_code         out int
    , in_dbms_logging      in int default 0
    , in_drop_temp_tables  in int default 1
    );
```

The following table describes the definition of the MDM_STG_PERSISTID_RETENTION.EXEC_STG_PERSISTID_DIRECT procedure:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_stage_table_name</td>
<td>VARCHAR2(30)</td>
<td>Stage table name. Input. Required.</td>
<td>Contains the stage table name.</td>
</tr>
</tbody>
</table>
### MDM Persist ID User Exits Implementation

To implement the Persistent ID Module, you must complete the following steps:

1. Identify or create the column to hold the Persistent ID.
2. Create the configuration and log tables.
3. Register the unique ID column.
4. Create user exit implementations.
5. Compile and export the user exit JAR file.
6. Deploy the user exit JAR file.

#### Step 1. Identify or Create the Column to Hold the Persistent ID

To create or identify the MDM base object column that contains the unique ID value, perform the following steps:

1. In the Hub Console, create or identify the MDM Hub base object column that contains the unique ID value. This column must be of a character data type.
2. Enable the **Trust and Validation** option on the column.
3. Create or enable the stage table column to include the Persistent ID column for all data sources.
4. Create or enable the landing to staging table mapping to include the Persistent ID column for all data sources.
5. Set the minimum and maximum trust values to 80 for the Persistent ID column across all source systems.
   
   **Note:** You can use other trust values, but the minimum and maximum trust values must be the same. They must also have a value of 5 or greater.

#### Step 2. Create the Configuration and Log Tables

After you create the unique ID column, you must create and register the configuration and log tables for the Persistent ID Module. To create the configuration and log tables, see the sample scripts in the appendixes of this document.

**RELATED TOPICS:**

- "Appendix 1. Oracle SQL Script for Persistent ID Setup" on page 25
- "Appendix 2. MS SQL Server SQL Script for Persistent ID Setup" on page 27
Step 3. Register the Unique ID Column

After you setup the configuration and log tables, register the unique ID column in the C_REPOS_PERSIST_COLUMN_CONFIG table.

You can refer to the examples of a unique ID registration for Oracle and Microsoft SQL Server that are provided.

Oracle Unique ID Registration Example

The following Oracle insert query registers the UNIQUE_ID column in the C_BO_PERSIST_ID base object in the C_REPOS_PERSIST_COLUMN_CONFIG table.

```sql
INSERT INTO C_REPOS_PERSIST_COLUMN_CONFIG
( ROWID_PERSIST_COLUMN
  , ROWID_COLUMN
  , ENABLED_IND
  , UNIQUE_ID_TYPE
  , UNIQUE_ID_FORMAT_FUNCTION
  , ALLOW_UNMERGE_XREF_WITH_ID
  , ALLOW_DELETE_XREF_WITH_ID
  , IGNORE_XREF_HSI
  , IGNORE_XREF_TIMELINE
  , DOWNGRADE_LOSING_XREF_PERCENT
  , SURVIVORSHIP_TYPE
  , MERGE_LIST_EVAL_SIZE
  , JOB_METRIC_TYPE_CODE
  , UNMERGE_REJECT_MSG )
select
  ( 'CPID.' || to_char( nvl((select max(substr(ROWID_PERSIST_COLUMN, 6,9) + 1)
                         from c_repos_persist_column_config), 1), 'FM0000')
      ) ROWID_PERSIST_COLUMN
, ROWID_COLUMN
, 1 ENABLED_IND
-- example Is using Unique ID Type of 4 with a formatted rowid_xref - Prefixed by PA and 12 digits
, 3 UNIQUE_ID_TYPE
, ''PA'' || ' || TO_CHAR(ROWID_XREF, ''FM000000000000'')' UNIQUE_ID_FORMAT_FUNCTION
, 0 ALLOW_UNMERGE_XREF_WITH_ID
, 0 ALLOW_DELETE_XREF_WITH_ID
, 0 IGNORE_XREF_HSI
, 0 IGNORE_XREF_TIMELINE
, 90 DOWNGRADE_LOSING_XREF_PERCENT
, 0 SURVIVORSHIP_TYPE             -- from oldest rowid_xref
, 300 MERGE_LIST_EVAL_SIZE
, 101 JOB_METRIC_TYPE_CODE
, ('The requested Cross Reference Record cannot be unmerged, it contains the Persistent Identifier for Base Object Record with ROWID_OBJECT of "${ROWID_OBJECT}".') UNMERGE_REJECT_MSG
from c_repos_column
where table_name = 'C_BO_PERSIST_ID'       --- THE BASE OBJECT TABLE NAME
and column_name = 'UNIQUE_ID';            --- THE UNIQUE ID COLUMN NAME
commit;
```

Microsoft SQL Server Unique ID Registration Example

The following Microsoft SQL Server insert query registers the UNIQUE_ID column in the C_BO_PERSIST_ID base object in the C_REPOS_PERSIST_COLUMN_CONFIG table.

```sql
-- configure a PERSIST ID COLUMN
BEGIN TRANSACTION
INSERT INTO C_REPOS_PERSIST_COLUMN_CONFIG
( ROWID_PERSIST_COLUMN
  , ROWID_COLUMN
  , ENABLED_IND
  , UNIQUE_ID_TYPE
) select
  ( 'CPID.' || to_char( nvl((select max(substr(ROWID_PERSIST_COLUMN, 6,9) + 1)
                         from c_repos_persist_column_config), 1), 'FM0000')
      ) ROWID_PERSIST_COLUMN
, ROWID_COLUMN
, 1 ENABLED_IND
-- example Is using Unique ID Type of 4 with a formatted rowid_xref - Prefixed by PA and 12 digits
, 3 UNIQUE_ID_TYPE
, ''PA'' || ' || TO_CHAR(ROWID_XREF, ''FM000000000000'')' UNIQUE_ID_FORMAT_FUNCTION
, 0 ALLOW_UNMERGE_XREF_WITH_ID
, 0 ALLOW_DELETE_XREF_WITH_ID
, 0 IGNORE_XREF_HSI
, 0 IGNORE_XREF_TIMELINE
, 90 DOWNGRADE_LOSING_XREF_PERCENT
, 0 SURVIVORSHIP_TYPE             -- from oldest rowid_xref
, 300 MERGE_LIST_EVAL_SIZE
, 101 JOB_METRIC_TYPE_CODE
, ('The requested Cross Reference Record cannot be unmerged, it contains the Persistent Identifier for Base Object Record with ROWID_OBJECT of "$(ROWID_OBJECT)".') UNMERGE_REJECT_MSG
from c_repos_column
where table_name = 'C_BO_PERSIST_ID'       --- THE BASE OBJECT TABLE NAME
and column_name = 'UNIQUE_ID';            --- THE UNIQUE ID COLUMN NAME
commit;
```
Step 4. Create User Exit Implementations

To create user exit implementations, you can use Eclipse or another development environment. For example, you create user exit implementations in an Eclipse project called MDMUserExitImplementation.

1. In Eclipse, right-click the Project Explorer panel and select New > Project.
2. In the Create Project Wizard, select Java Project. Click Next.
3. For the project name, type MDMUserExitImplementation and click Finish.
4. Navigate to the directory where the MDMUserExitImplementation folder is located. In the MDMUserExitImplementation folder, create a folder called lib.
5. From the <MDM installation directory>/server/hub/lib directory, copy the following files to the lib folder:
   - log4j-1.2.16.jar
   - mdm-ue.jar
6. From the directory where you saved it, copy the MDMPersistIdUserExit_<mdm_version#>.jar file to the lib folder.
7. Extract the MDMPersistIdUserExit_<mdm_version#>.jar file to the root of the Eclipse project directory.
The following directory and subdirectories appear: \com\informatica\mdm\ue\persistid. The persistid subdirectory contains all the .class files that are needed to create the user exit JAR for MDM user exit implementation.

8. In Eclipse, right-click **MDMUserExitImplementation** and select **Refresh**.
   The lib folder that contains the JAR files appears.

9. Right-click each JAR file, and select **Build Path > Add to Build Path**.
   The JAR files appear in the Referenced Library folder.

10. To create a package, right-click the **src** folder and select **New > Package**.

11. For the package name, type **com.<the_company_name>.mdm.userexits** and click **Finish**.
    Where `<the_company_name>` is the name of your company.

12. Under the package, create the following Java classes to implement the corresponding MDM user exits:
    - PostStage
    - PostLoad
    - PostMerge
    - PostUnmerge
    - PreUnmerge

13. In each Java class, add user exit code to invoke Persistent ID processing. For each Java class, refer to the following code samples:

    Note: When you add the sample code to a Java class, change `<THE_COMPANY_NAME>` to match the name that you used in the `com.<the_company_name>.mdm.userexits` package.

**PostLoad Class to Invoke the Persistent ID Processing**

To create the PostLoad class to invoke the persistent ID processing, use the following code:

```java
package com.<THE_COMPANY_NAME>.mdm.userexits;
import java.util.*;
import org.apache.log4j.Logger;
import com.informatica.mdm.api.put.ActionType;
import com.informatica.mdm.userexit.PostLoadUserExit;
import com.informatica.mdm.userexit.UserExitContext;
import com.informatica.mdm.ue.persistid.PersistId;
import com.informatica.mdm.ue.persistid.PersistIdExceptions;

public class PostLoad implements PostLoadUserExit
{
    private static final Logger log = Logger.getLogger(PostLoad.class.getName());
    private static final String _simpleClassName = PostLoad.class.getSimpleName();

    @Override
    public void processUserExit( UserExitContext userExitContext, ActionType actionType,
        Map<String, Object> boDataMap, Map<String, Object> xrefDataMap,
        List<Map<String, Object>> xrefDataMapList ) throws Exception
    {
        // START of POST_LOAD PERSIST ID processing
        log.debug( "Invoking " + PersistId.class.getSimpleName() );
        long startTime = System.currentTimeMillis();    // start time in milliseconds
        PersistId persistId = new PersistId( PersistId.UserExitType.POST_LOAD, userExitContext,
            actionType,
            boDataMap, xrefDataMap, xrefDataMapList);
    }

```

19
try {
    persistId.executePersistId();
} catch (PersistIdExceptions pidEx) {
    log.warn( PersistId.class.getSimpleName() + " Return Code: " + pidEx.getReturnCode() + " Error Message: " + pidEx.getMessage());
    long endTime = System.currentTimeMillis(); //end time in milliseconds
    log.error( "Aborting " + PersistId.class.getSimpleName() + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " + persistId.getReturnMessage());
    log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms");
} catch (Exception ex) {
    log.error( "Aborting " + PersistId.class.getSimpleName() + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " + persistId.getReturnMessage());
    long endTime = System.currentTimeMillis(); //end time in milliseconds
    log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms on rowidObject " + boDataMap.get("ROWID_OBJECT") + ");
    throw new Exception(ex);
} catch (Exception ex) {
    log.error( "Aborting " + PersistId.class.getSimpleName() + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " + persistId.getReturnMessage());
    long endTime = System.currentTimeMillis(); //end time in milliseconds
    log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms on rowidObject " + boDataMap.get("ROWID_OBJECT") + ");
    throw new Exception(ex);
}

log.debug("Completed " + _simpleClassName + " Return Code: " + persistId.getReturnCode() + " Error Message: " + persistId.getReturnMessage());
long endTime = System.currentTimeMillis(); //end time in milliseconds
log.debug(_simpleClassName + " PersistId Completed in: " + (endTime - startTime) + " ms on rowidObject " + boDataMap.get("ROWID_OBJECT") + ");
**************************************************************************
}

PostMerge Class to Invoke the Persistent ID Processing

To create the PostMerge class to invoke the persistent ID processing, use the following code:

```java
package com.<THE_COMPANY_NAME>.mdm.userexits;
import java.util.*;
import org.apache.log4j.Logger;
import com.informatica.mdm.ue.persistid.PersistId;
import com.informatica.mdm.ue.persistid.PersistIdExceptions;
import com.informatica.mdm.userexit.PostMergeUserExit;
import com.informatica.mdm.userexit.UserExitContext;

public class PostMerge implements PostMergeUserExit {
    private static final Logger log = Logger.getLogger(PostMerge.class.getName());
    private static final String _simpleClassName = PostMerge.class.getSimpleName();

    @Override
    public void processUserExit( UserExitContext userExitContext, Map<String, List<String>> rowidObjectMapList )
        throws Exception {
        try {
            PersistId persistId = new PersistId(PersistId.UserExitType.POST_MERGE, userExitContext, rowidObjectMapList);
            try {
                persistId.executePersistId();
            } catch (PersistIdExceptions pidEx) {
                log.warn( PersistId.class.getSimpleName() + " Return Code: " + pidEx.getReturnCode() + " Error Message: " + pidEx.getMessage());
            }
        } catch (Exception ex) {
            log.error( "Aborting " + PersistId.class.getSimpleName() + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " + persistId.getReturnMessage());
            long endTime = System.currentTimeMillis(); //end time in milliseconds
            log.debug(_simpleClassName + " PersistId Completed in: " + (endTime - startTime) + " ms on rowidObject " + boDataMap.get("ROWID_OBJECT") + ");
            throw new Exception(ex);
        }
    }
}
```

```”}

PostMerge Class to Invoke the Persistent ID Processing

To create the PostMerge class to invoke the persistent ID processing, use the following code:

```java
package com.<THE_COMPANY_NAME>.mdm.userexits;
import java.util.*;
import org.apache.log4j.Logger;
import com.informatica.mdm.ue.persistid.PersistId;
import com.informatica.mdm.ue.persistid.PersistIdExceptions;
import com.informatica.mdm.userexit.PostMergeUserExit;
import com.informatica.mdm.userexit.UserExitContext;

public class PostMerge implements PostMergeUserExit {
    private static final Logger log = Logger.getLogger(PostMerge.class.getName());
    private static final String _simpleClassName = PostMerge.class.getSimpleName();

    @Override
    public void processUserExit( UserExitContext userExitContext, Map<String, List<String>> rowidObjectMapList )
        throws Exception {
        try {
            PersistId persistId = new PersistId(PersistId.UserExitType.POST_MERGE, userExitContext, rowidObjectMapList);
            try {
                persistId.executePersistId();
            } catch (PersistIdExceptions pidEx) {
                log.warn( PersistId.class.getSimpleName() + " Return Code: " + pidEx.getReturnCode() + " Error Message: " + pidEx.getMessage());
            }
        } catch (Exception ex) {
            log.error( "Aborting " + PersistId.class.getSimpleName() + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " + persistId.getReturnMessage());
            long endTime = System.currentTimeMillis(); //end time in milliseconds
            log.debug(_simpleClassName + " PersistId Completed in: " + (endTime - startTime) + " ms on rowidObject " + boDataMap.get("ROWID_OBJECT") + ");
            throw new Exception(ex);
        }
    }
}
```
PreUnmerge Class to Invoke the Persistent ID Processing

To create the PreUnmerge class to invoke the persistent ID processing, use the following code:

```java
package com.<THE_COMPANY_NAME>.mdm.userexits;

import java.util.*;
import org.apache.log4j.Logger;
import com.informatica.mdm.ue.persistid.PersistId;
import com.informatica.mdm.ue.persistid.PersistIdExceptions;
import com.informatica.mdm.userexit.PreUnmergeUserExit;
import com.informatica.mdm.userexit.UnmergeKey;
import com.informatica.mdm.userexit.UserExitContext;

public class PreUnmerge implements PreUnmergeUserExit
{
    private static final Logger log = Logger.getLogger(PreUnmerge.class.getName());
    private static final String _simpleClassName = PreUnmerge.class.getSimpleName();

    @Override
    public void processUserExit( UserExitContext userExitContext, Set<UnmergeKey> unmergeKey)
    throws Exception
    {
        // START of PRE_UNMERGE PERSIST ID processing
       **************************************************************************
        log.debug( "Invoking " + PersistId.class.getSimpleName() );
        long startTime = System.currentTimeMillis(); //start time in milliseconds
        PersistId persistId = new PersistId(PersistId.UserExitType.PRE_UNMERGE, userExitContext, unmergeKey);
        try {
            persistId.executePersistId();
        } catch (PersistIdExceptions pidEx) {
            log.warn( PersistId.class.getSimpleName() + " Return Code: " + pidEx.getReturnCode() + " Error Message: " + pidEx.getMessage());
            long endTime = System.currentTimeMillis(); //end time in milliseconds
            log.error( "Aborting " + PersistId.class.getSimpleName() );
            throw new Exception(pidEx.getMessage());
        } catch (Exception ex) {
            log.error( "Aborting " + PersistId.class.getSimpleName() + " Return Code: " + persistId.getReturnCode() + " Return Message: " + persistId.getReturnMessage());
            long endTime = System.currentTimeMillis(); //end time in milliseconds
            log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms");
            throw new Exception(ex);
        }
        log.debug("Completed " + _simpleClassName + " Return Code: " + persistId.getReturnCode() + "; Error Message: " + persistId.getReturnMessage());
        long endTime = System.currentTimeMillis(); //end time in milliseconds
        log.debug(_simpleClassName + " PersistId Completed in: " + (endTime - startTime) + " ms" + " on merged rowid_object targets of " + rowidObjectMapList.size());
        // END of POST_MERGE PERSIST ID processing
        **************************************************************************
    }
}
```
throw new Exception(pidEx.getMessage());
} catch (Exception ex) {
    log.error("Aborting " + PersistId.class.getSimpleName()
        + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " +
        persistId.getReturnMessage());
    long endTime = System.currentTimeMillis();  // end time in milliseconds
    log.debug(_simpleClassName  + " PersistId run for: " + (endTime - startTime) + " ms");
    throw new Exception(ex);
}

log.debug("Completed " + _simpleClassName
        + " Return Code: " + persistId.getReturnCode() + "; Error Message: " +
        persistId.getReturnMessage());
long endTime = System.currentTimeMillis();  // end time in milliseconds
log.debug(_simpleClassName  + " PersistId Completed in " + (endTime - startTime) + " ms" + "+ " + on UnmergeKeySet Count of " + unmergeKey.size());
// END of PRE_UNMERGE PERSIST ID processing
**************************************************************************

PostUnmerge Class to Invoke the Persistent ID Processing

To create the PostUnmerge class to invoke the persistent ID processing, use the following code:

```java
package com.<THE_COMPANY_NAME>.mdm.userexits;

import java.util.*;
import org.apache.log4j.Logger;
import com.informatica.mdm.ue.persistid.PersistId;
import com.informatica.mdm.ue.persistid.PersistIdExceptions;
import com.informatica.mdm.userexit.PostUnmergeUserExit;
import com.informatica.mdm.userexit.PostUnmergeResponse;
import com.informatica.mdm.userexit.UserExitContext;

public class PostUnmerge implements PostUnmergeUserExit {
    private static final Logger log = Logger.getLogger(PostUnmerge.class.getName());
    private static final String _simpleClassName = PostUnmerge.class.getSimpleName();

    @Override
    public void processUserExit(UserExitContext userExitContext, Set<PostUnmergeResponse>
        postUnmergeResponse) throws Exception {

        // START of POST_UNMERGE PERSIST ID processing
        log.debug("Invoking " + PersistId.class.getSimpleName() );
        Long startTime = System.currentTimeMillis();  // start time in milliseconds

        PersistId persistId = new PersistId(PersistId.UserExitType.POST_UNMERGE, userExitContext,
            postUnmergeResponse, postUnmergeResponse.size() );

        try {
            persistId.executePersistId();
        } catch (PersistIdExceptions pidEx) {
            log.warn(" Aborting " + PersistId.class.getSimpleName()
                + " Return Code: " + pidEx.getReturnCode() + "; Error Message: " +
                pidEx.getMessage());
            log.error(" Aborting " + PersistId.class.getSimpleName() );
            long endTime = System.currentTimeMillis();  // end time in milliseconds
            log.debug(_simpleClassName  + " PersistId run for: " + (endTime - startTime) + " ms");
            throw new Exception(pidEx.getMessage());
        } catch (Exception ex) {
            log.error(" Aborting " + PersistId class.getSimpleName() + "; Return Code: " + persistId.getReturnCode() + "; Return Message: " +
```

22
PersistId.getReturnMessage();
        long endTime = System.currentTimeMillis();    //end time in milliseconds
        log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms");
        throw new Exception(ex);
    }

    log.debug( "Completed " + _simpleClassName + " Return Code: " + persistId.getReturnCode() + " Error Message: " + persistId.getReturnMessage());
    Long endTime = System.currentTimeMillis();    //end time in milliseconds
    log.debug(_simpleClassName + " PersistId Completed in " + (endTime - startTime) + " ms" + " on PostUnmergeResponse count of " + postUnmergeResponse.size());
// END of POST_UNMERGE PERSIST ID processing
***********************************************************************
 }
}

PostStage Class to Invoke the Persistent ID Processing

To create the PostStage class to invoke the persistent ID processing, use the following code:

```java
package com.<THE_COMPANY_NAME>.mdm.userexits;
import org.apache.log4j.Logger;
import com.informatica.mdm.ue.persistid.PersistId;
import com.informatica.mdm.ue.persistid.PersistIdExceptions;
import com.informatica.mdm.userexit.PostStageUserExit;
import com.informatica.mdm.userexit.UserExitContext;

public class PostStage implements PostStageUserExit
{
    private static final Logger log = Logger.getLogger(PostStage.class.getName());
    private static final String _simpleClassName = PostStage.class.getSimpleName();

    @Override
    public void processUserExit( UserExitContext userExitContext , String stageTableName , String landingTableName , String previousLandingTableName ) throws Exception
    {
        // START of POST_STAGE PERSIST ID processing ********************************
        log.info( "Invoking " + PersistId.class.getSimpleName() );
        long startTime = System.currentTimeMillis();    //start time in milliseconds
        PersistId persistId = new PersistId( PersistId.UserExitType.POST_STAGE, userExitContext , stageTableName , landingTableName , previousLandingTableName );
        try {
            persistId.executePersistId();
        } catch (PersistIdExceptions pidEx) {
            log.warn( PersistId.class.getSimpleName() + " Return Code: " + pidEx.getReturnCode() + " Error Message: " + pidEx.getMessage());
            log.error( "Aborting " + PersistId.class.getSimpleName() );
            long endTime = System.currentTimeMillis();    //end time in milliseconds
            log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms");
            throw new Exception(pidEx.getMessage());
        } catch (Exception ex) {
            log.error( "Aborting " + PersistId.class.getSimpleName() + " Return Code: " + persistId.getReturnCode() + " Return Message: " + persistId.getReturnMessage() );
            long endTime = System.currentTimeMillis();    //end time in milliseconds
            log.debug(_simpleClassName + " PersistId run for: " + (endTime - startTime) + " ms");
            throw new Exception(ex);
        }
```

23
log.debug("Completed " + _simpleClassName + " Return Code: " + persistId.getReturnCode() + " Error Message: " + persistId.getReturnMessage());
long endTime = System.currentTimeMillis(); // end time in milliseconds
log.debug(_simpleClassName + " PersistId Completed in: " + (endTime - startTime) + " ms on stage table " + stageTableName);
// END of POST_STAGE PERSIST ID processing *******************************

Step 5. Compile and Export the User Exit JAR File

After you create the user exit classes, compile the project and export the user exit JAR file for MDM user exit implementation.

The user exit implementation JAR file must include the user exit class file and the MDM Persistent ID module class file. When you export the JAR file from Eclipse, select the check box of the root directory where you extracted the MDM Persistent ID class file. The following image shows an example of the directories to select to export class files and resources:

![Image of JAR File Specification]

Step 6. Deploy the User Exit JAR File

To deploy user exits, you register the user exits in the ORS through the Hub Console.

1. Acquire a write lock.
2. From the Hub Console, select User Object Registry > User Exits.
3. Click Add in the upper right.
4. Browse to the location of the user exit JAR file.
5. Click OK.
**Hub Server and Process Server Logging Configuration**

The MDM Persistent ID user exits sends messages and errors to the log4j.xml file.

The log4j.xml file is in the following directory:

```
<MDM Hub installation directory>/hub/server/conf
```

To control the logging, both the Hub Server and Process Server log4j.xml file must include the `com.informatica.mdm.ue.persistid` logging entry for the MDM Persistent ID.

For example:

```
<category name="com.informatica.mdm.ue.persistid">
    <priority value="INFO"/>    <!--CAN BE SET TO DEBUG FOR DEBUGGING -->
</category>
```

If you set the value to DEBUG, the logging will be extremely high during the MDM Load, Merge, and Unmerge processes. Informatica recommends to leave the logging mode in INFO mode and only set to DEBUG if required. If you change the the logging mode, you do not need to restart the application server.

**Deadlock Issue in a Microsoft SQL Server Environment**

In a Microsoft SQL Server environment, a deadlock error might occur in the Load and Merge processes. The issue occurs during the synchronization of the _HIST and _HXRF tables on the base object that contains the unique ID to persist.

If a deadlock issue occurs, the index on the ROWID_OBJECT column in the _HIST table must be dropped and recreated as a clustered index. The index on the ROWID_XREF on the _HXRF HIST table must also be dropped and recreated as a clustered index.

**Appendix 1. Oracle SQL Script for Persistent ID Setup**

To set up the Persistent ID module in an Oracle environment, you can use a prepared SQL script.

The following script performs the following actions:

- Creates the `C_REPOS_PERSIST_COLUMN_CONFIG` table.
- Creates the index on the ROWID_COLUMN.
- Registers the `C_REPOS_PERSIST_COLUMN_CONFIG` table with the columns.
- Creates the `C_LOG_RESET_DLT_TO_ACTV` table.

Note: After you run the script, the MDM Repository Manager displays warning messages about system table `C_REPOS_PERSIST_COLUMN_CONFIG` and `UI_PERSIST_COL_ROWID_COLUMN` affecting the metadata. You can ignore these warning messages.

```
-- CREATE THE C_REPOS_PERSIST_COLUMN_CONFIG TABLE
CREATE TABLE C_REPOS_PERSIST_COLUMN_CONFIG
(
    ROWID_PERSIST_COLUMN CHAR(14 CHAR) NOT NULL,
    ROWID_COLUMN CHAR(14 CHAR) NOT NULL,
    ENABLED_IND INTEGER DEFAULT 1 NOT NULL,
    SURVIVORSHIP_TYPE INTEGER DEFAULT 0 NOT NULL,
    UNIQUE_ID_TYPE INTEGER DEFAULT 0 NOT NULL,
    UNIQUE_ID_FORMAT_FUNCTION VARCHAR2(255 CHAR),
    ALLOW_UNMERGE_XREF_WITH_ID INTEGER DEFAULT 0 NOT NULL,
    ALLOW_DELETE_XREF_WITH_ID INTEGER DEFAULT 0 NOT NULL,
    IGNORE_XREF_HSI INTEGER DEFAULT 0 NOT NULL,
    IGNORE_XREF_TIMELINE INTEGER DEFAULT 0 NOT NULL,
    DOWNGRADE_LOSING_XREF_PERCENT INTEGER DEFAULT 90 NOT NULL,
    JOB_METRIC_TYPE_CODE INTEGER DEFAULT 101 NOT NULL,
    UNMERGE_REJECT_MSG VARCHAR2(1024 CHAR) NOT NULL
);
```
-- ADD THE CONSTRAINT CHECKS TO THE C_REPOS_PERSIST_COLUMN_CONFIG TABLE
ALTER TABLE C_REPOS_PERSIST_COLUMN_CONFIG ADD (
    CHECK ( ENABLED_IND  in (0,1) ) ENABLE VALIDATE,
    CHECK ( UNIQUE_ID_TYPE in (0,1,2,3) ) ENABLE VALIDATE,
    CHECK (ALLOW_UNMERGE_XREF_WITH_ID IN (0,1)) ENABLE VALIDATE,
    CHECK (ALLOW_DELETE_XREF_WITH_ID IN (0,1) ) ENABLE VALIDATE,
    CHECK ( IGNORE_XREF_HSI  in (0,1) ) ENABLE VALIDATE,
    CHECK ( IGNORE_XREF_TIMELINE  in (0,1) ) ENABLE VALIDATE,
    CHECK ( DOWNGRADE_LOSING_XREF_PERCENT between 1 and 100 ) ENABLE VALIDATE,
    CHECK ( SURVIVORSHIP_TYPE IN (0,1) ) ENABLE VALIDATE);

-- CREATE THE INDEX ON THE ROWID_COLUMN
CREATE INDEX UI_PERSIST_COL_ROWID_COLUMN ON C_REPOS_PERSIST_COLUMN_CONFIG
(ROWID_COLUMN)
NOLOGGING TABLESPACE CMX_INDX NOPARALLEL;

-- REGISTER THE JOB METRIC TYPE
INSERT INTO C_REPOS_JOB_METRIC_TYPE
( METRIC_TYPE_CODE, CREATE_DATE, CREATOR, LAST_UPDATE_DATE, UPDATED_BY,
  METRIC_TYPE_DESC, SEQ ) VALUES
( 101, SYSTIMESTAMP, 'PersistID', SYSTIMESTAMP, 'PersistID',
  'Records reset from Delete to Active', 101 ) ;
COMMIT;

--- CREATE THE STAGE JOB DELETE TO ACTIVE RESET LOG TABLE
CREATE TABLE C_LOG_RESET_DLT_TO_ACTV
( CREATE_DATE             TIMESTAMP(6),
  ROWID_JOB_CNTRL         CHAR(14 CHAR),
  STG_TABLE_NAME          VARCHAR2(50 CHAR),
  XREF_TABLE_NAME         VARCHAR2(50 CHAR),
  STG_SRC_ROWID           VARCHAR2(30 CHAR),
  STG_LAST_UPDATE_DATE    TIMESTAMP(6),
  STG_PKEY_SRC_OBJECT     VARCHAR2(255 CHAR),
  STG_HUB_STATE_IND       INTEGER,
  STG_VERSION_SEQ         INTEGER,
  STG_PERIOD_START_DATE   TIMESTAMP(6),
  STG_PERIOD_END_DATE     TIMESTAMP(6),
  STG_PERSIST_ID_VALUE    VARCHAR2(255 CHAR),
  XREF_ROWID_XREF         INTEGER,
  XREF_ROWID_OBJECT       CHAR(14 CHAR),
  XREF_PKEY_SRC_OBJECT    VARCHAR2(255 CHAR),
  XREF_ROWID_SYSTEM       CHAR(14 CHAR),
  XREF_HUB_STATE_IND      INTEGER,
  XREF_PERIOD_START_DATE  TIMESTAMP(6),
  XREF_PERIOD_END_DATE    TIMESTAMP(6),
  XREF_PERSIST_ID_VALUE   VARCHAR2(255 CHAR)
) TABLESPACE CMX_DATA
Appendix 2. MS SQL Server SQL Script for Persistent ID Setup

To set up the Persistent ID module in an Microsoft SQL Server environment, you can use a prepared SQL script. The following script performs the following actions:

• Creates the `C_REPOS_PERSIST_COLUMN_CONFIG` table.
• Creates the index on the `ROWID_COLUMN`.
• Registers the `C_REPOS_PERSIST_COLUMN_CONFIG` table with the columns.
• Creates the `C_LOG_RESET_DLT_TO_ACTV` table.

Note: After you run the script, the MDM Repository Manager displays warning messages about system table `C_REPOS_PERSIST_COLUMN_CONFIG` and UI_PERSIST_COL_ROWID_COLUMN affecting the metadata. You can ignore these warning messages.

```sql
USE CMX_ORS
GO

-- CREATE THE C_REPOS_PERSIST_COLUMN_CONFIG TABLE
CREATE TABLE C_REPOS_PERSIST_COLUMN_CONFIG
(
  ROWID_PERSIST_COLUMN          NCHAR (14)          NOT NULL,
  ROWID_COLUMN                  NCHAR (14)          NOT NULL,
  ENABLED_IND                   INT                 DEFAULT 1   NOT NULL,
  SURVIVORSHIP_TYPE             INT                 DEFAULT 0   NOT NULL,
  UNIQUE_ID_TYPE                INT                 DEFAULT 0   NOT NULL,
  UNIQUE_ID_FORMAT_FUNCTION     NVARCHAR (255),
  ALLOW_UNMERGE_XREF_WITH_ID    INT                 DEFAULT 0   NOT NULL,
  ALLOW_DELETE_XREF_WITH_ID     INT                 DEFAULT 0   NOT NULL,
  IGNORE_XREF_HSI               INT                 DEFAULT 0   NOT NULL,
  IGNORE_XREF_TIMELINE          INT                 DEFAULT 0   NOT NULL,
  DOWNGRADE_LOSING_XREF_PERCENT INT                 DEFAULT 90  NOT NULL,
  JOB_METRIC_TYPE_CODE          INT                 DEFAULT 101 NOT NULL,
  UNMERGE_REJECT_MSG            NVARCHAR(1024),
  MERGE_LIST_EVAL_SIZE          INT                 DEFAULT 300 NOT NULL,
  PRECEDENCE_SOURCE_SYSTEMS     NVARCHAR (255),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_ENABLED_IND CHECK (ENABLED_IND=1 OR ENABLED_IND=0),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_UNIQUE_ID_TYPE CHECK (UNIQUE_ID_TYPE=3 OR UNIQUE_ID_TYPE=2 OR UNIQUE_ID_TYPE=1 OR UNIQUE_ID_TYPE=0),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_ALLOW_UNMERGE_XREF_WITH_ID CHECK (ALLOW_UNMERGE_XREF_WITH_ID=1 OR ALLOW_UNMERGE_XREF_WITH_ID=0),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_ALLOW_DELETE_XREF_WITH_ID CHECK (ALLOW_DELETE_XREF_WITH_ID=1 OR ALLOW_DELETE_XREF_WITH_ID=0),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_IGNORE_XREF_HSI CHECK (IGNORE_XREF_HSI=1 OR IGNORE_XREF_HSI=0),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_IGNORE_XREF_TIMELINE CHECK (IGNORE_XREF_TIMELINE=1 OR IGNORE_XREF_TIMELINE=0),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_DOWNGRADE_LOSING_XREF_PERCENT CHECK (DOWNGRADE_LOSING_XREF_PERCENT>=1 AND DOWNGRADE_LOSING_XREF_PERCENT<=100),
  CONSTRAINT CK_VAL_PERSIST_COLUMN_CONFIG_SURVIVORSHIP_TYPE CHECK (SURVIVORSHIP_TYPE=1 OR SURVIVORSHIP_TYPE=0)
) ON CMX_DATA;

-- ADD THE CONSTRAINT CHECKS TO THE C_REPOS_PERSIST_COLUMN_CONFIG TABLE
CREATE UNIQUE NONCLUSTERED INDEX UI_PERSIST_COL_ROWID_COLUMN
  ON C_REPOS_PERSIST_COLUMN_CONFIG (ROWID_COLUMN) ON CMX_INDEX;

GO

-- add a JOB METRIC for stage process reset to Active from Deleted
BEGIN TRANSACTION

--- CREATE THE STAGE JOB METRIC FOR RESETTING REJECTS TO ACTIVES
```
INSERT INTO C_REPOS_JOB_METRIC_TYPE
(METRIC_TYPE_CODE, CREATE_DATE, CREATOR, LAST_UPDATE_DATE, UPDATED_BY, METRIC_TYPE_DESC, SEQ)
VALUES
(101, CURRENT_TIMESTAMP, 'PersistID', CURRENT_TIMESTAMP, 'PersistID', 'Records reset from Delete to Active', 101)

COMMIT;
GO

CREATE TABLE C_LOG_RESET_DLT_TO_ACTV
(
CREATE_DATE            DATETIME2 (7)  NULL,
ROWID_JOB_CNTRL        NCHAR (14)     NULL,
STG_TABLE_NAME         NVARCHAR (50)  NULL,
XREF_TABLE_NAME        NVARCHAR (50)  NULL,
STG_SRC_ROWID          NVARCHAR (30)  NULL,
STG_LAST_UPDATE_DATE   DATETIME2 (7)  NULL,
STG_PKEY_SRC_OBJECT    NVARCHAR (255) NULL,
STG_HUB_STATE_IND      BIGINT         NULL,
STG_VERSION_SEQ        BIGINT         NULL,
STG_PERIOD_START_DATE  DATETIME2 (7)  NULL,
STG_PERIOD_END_DATE    DATETIME2 (7)  NULL,
STG_PERSIST_ID_VALUE   NVARCHAR (255) NULL,
XREF_ROWID_XREF        BIGINT         NULL,
XREF_ROWID_OBJECT      NCHAR (14)     NULL,
XREF_PKEY_SRC_OBJECT   NVARCHAR (255) NULL,
XREF_ROWID_SYSTEM      NCHAR (14)     NULL,
XREF_HUB_STATE_IND     BIGINT         NULL,
XREF_PERIOD_START_DATE DATETIME2 (7)  NULL,
XREF_PERIOD_END_DATE   DATETIME2 (7)  NULL,
XREF_PERSIST_ID_VALUE  NVARCHAR (255) NULL,
) ON CMX_DATA;
GO

Author

Dan Kellett
The author would like to acknowledge Chuck Goon for his contribution to this article.