Oracle® Communications Billing and Revenue Management

Migrating Accounts to the BRM Database

Release 15.2
G35873-01
January 2026

ORACLE

Oracle Communications Billing and Revenue Management Migrating Accounts to the BRM Database, Release 15.2

G35873-01

Contents

3 Mapping Legacy Data to the BRM Data Schema

4 Migrating Data by Using New and Extended Storable Classes

5 Loading Legacy Data into the BRM Database

About This Content

This book describes how to migrate accounts from a legacy non-BRM database to a BRM database.

Audience

This book is for system database administrators and system administrators.

1

Understanding Conversion Manager

This chapter describes how to use Oracle Communications Billing and Revenue Management (BRM) Conversion Manager to convert data from a legacy database to the BRM database.

Topics in this document:

About Conversion Manager

You use Conversion Manager to load legacy data into BRM. Conversion Manager supports the following types of data:

- Account data, such as the customer's name, address, profile, and discount information.
- Service data, such as the services subscribed to by the accounts.
- Charge offer data, such as charge offers purchased by the accounts.
- Billing data.
- Account hierarchy data.
- Balance data, including data for all balance elements. When data is migrated, rollover events can use the balance data to manage rollovers.

You can load account and balance data together, or you can load account data first and balance data second. All account data for an account must be loaded before loading balance data.

Before migrating data, do the following:

- Create a price list that applies to the migrated data before migrating account and balance data. Migrated data must reference the price list in the BRM database.
- Make sure that cycle arrears fees, billing-time discounts, and rollovers have been processed in the legacy system.

When the accounts are migrated, cycle forward fees and discount grants (for example, minutes) are applied. However, all delayed events are handled in the current bill.

> **Note**
>
> Auditing is not supported by Conversion Manager.

Overview of the Data Conversion Process

Converting legacy data includes these tasks:

- Understanding the data in the legacy system and deciding how to convert it to the BRM database.
- Mapping the data in the legacy database to the BRM database. To do so, you create XML files, which are validated by the Conversion Manager XSD schema files.
- Migrating the data to the BRM database by using the **pin_cmt** utility. To migrate data:
  - Import data into the BRM database. The data is hidden from BRM processes.
  - Deploy the staged data to the production area.

  See "Loading Legacy Data into the BRM Database" for more information.

> **Note**
>
> BRM must be running when you import and deploy data.

About Testing Your Data Mapping

You test your data mapping by using a test database. To test the data mapping:

1. Create a subset of the data in an XML file. See "Mapping Legacy Data to the BRM Data Schema".
2. Load the data into the test database. See "Loading Legacy Data into the BRM Database".
3. Test the data. See "Testing the Imported Data".

After you determine that the data mapping works, you import the data into the production database.

About Mapping Data

You convert legacy data to XML format by using an extraction utility.

> **Note**
>
> Conversion Manager does not include a utility for extracting legacy data into XML files. You need to develop your own extraction utility or obtain it from third-party sources.

Conversion Manager converts data from any type of legacy database; for example, Oracle.

To map data from the legacy database to the BRM data schema, you need a thorough understanding of the legacy data and how BRM stores data.

If your legacy data includes data not currently supported in BRM, you can create new storable classes, or extend storable classes, to support the custom data. You can then extend the Conversion Manager XML schema to migrate the data. See "Migrating Data by Using New and Extended Storable Classes".

You can configure data enrichment to allow system-wide values to be bulk inserted and to provide batch controls for operational integrity.

About Loading Data

You use the **pin_cmt** utility to load the converted data into the BRM database. Loading data includes two processes: importing accounts and deploying accounts.

- First, you *import* the data into the BRM database. When the data is imported, it is hidden from BRM processes, so it is not part of your production system. (Data is hidden by changing the database number section in the Portal object ID (POID) of each imported object. The database number is set to a number defined in the **infranet.cmt.dbnumber** entry in the **Infranet.properties** file for the **pin_cmt** utility; for example, **0.0.0.12**.)

  When you import data, you specify a *stage ID*. This enables you to load data in stages; for example, you can load a specific type of account first. A single database schema can include multiple stages.

- After the data is imported, you *deploy* the data. When you deploy data, the imported data is exposed as production data.

  You control which accounts are deployed by entering the stage ID that you used when importing the data. In addition, you specify the billing day of month (DOM). Therefore, only those accounts with the specified stage ID and DOM are deployed.

  After the data is deployed, the database number in the POID is changed to the actual database number, and the data becomes available for use by BRM processes.

  > **Note**
  >
  > – Events that apply to a migrated account are *not* processed until the account is deployed.
  >
  > – When data is deployed, the database number used is defined in the **infranet.cmt.targetdbnumber** entry in the **pin_cmt** utility **Infranet.properties** file. By default, the number is **0.0.0.1**.
  >
  > – File processing data, such as the file name and batch ID, is stored in the **/batch/cmt** object.

  When accounts are deployed:

  – The bill cycles are started.

  – Cycle fees are applied.

  – If you use a multischema system, the uniqueness table in the primary database schema is updated.

About Verifying Data Before It Is Deployed

You can look at data that is imported but not yet deployed by logging in to Billing Care connected to a scoped Oracle Data Manager (DM). See "Viewing Data before Deploying".

About Migrating Data to Multischema Systems

To load data into a multischema system, run a separate instance of the **pin_cmt** utility for each database schema. The **pin_cmt** utility connects to the primary schema and creates a **/uniqueness** object for every account and service migrated to BRM.

> **Note**
>
> If you have a multischema system, make sure the stage IDs are larger than the database IDs for the schemas. For example, if you have a schema with the number 0.0.0.5, use stage IDs larger than 5.

About Loading Data by Using Multiple Files

You can use multiple XML files to load data. For example, you can load account information separately from balance data. You can use the same staging area for multiple files.

> **Note**
>
> - Load account data before loading balance data.
> - Load parent accounts before loading child accounts.

About Reloading Data

If the **pin_cmt** utility runs out of space in your BRM database for data rows, the loading process stops. Data that is not loaded can be loaded after more space is made available in the database. See "Reloading Data".

2

Installing and Configuring Conversion Manager

This chapter describes how to install Oracle Communications Billing and Revenue Management (BRM) Conversion Manager.

Topics in this document:

- Installing Conversion Manager
- Configuring the pin_cmt Utility
- Improving Conversion Manager Performance

See also "Understanding Conversion Manager".

Installing Conversion Manager

Conversion Manager is available for the Linux operating system and for the Oracle database.

> **Note**
>
> If you already installed the product, you must uninstall its features before reinstalling them.

Ensure that the **libocijdbc11.so** file is available in your system. Conversion Manager requires this OCI Instant JDBC Library.

To install Conversion Manager, see "Installing Individual BRM Components" in *BRM Installation Guide*.

Configuring the pin_cmt Utility

You use the **pin_cmt** utility to migrate the data. To configure performance, log file, and other settings for this utility, edit the **Infranet.properties** file in *BRM_home***/apps/cmt**.

> **Note**
>
> Many of the entries in the **Infranet.properties** file include default settings that are applicable for testing your data mapping; for example, the number of threads that the **pin_cmt** utility uses. You can change the settings when you load data into the production system.

In addition, you need to edit the **pin.conf** file for the **cmt_mta_cycle_fees** utility. This utility is run automatically by the **pin_cmt** utility to apply cycle fees. The **pin.conf** file is in *BRM_home***/**

apps/cmt. It includes the standard connection parameters and multithreaded application (MTA) performance setting. You can also specify to not apply cycle fees to accounts.

See "Applying Cycle Fees to Deployed Accounts".

Referencing JAR Files

To ensure that the **pin_cmt** utility finds all the necessary JAR files, include them in the **pin_cmt** file, in the line that invokes Java. The format is:

```
BRM_home/jre/bin/java -Dfile.encoding=utf-8 -cp "jar_A:jar_B:jar_C:jar_D"
com.portal.cmt.Cmt $1 $2 $3 $4 $5 $6 $7 $8 $9
```

where *jar_A*:*jar_B*:*jar_C*:*jar_D* is the list of Java class JAR files. Oracle library references are appended at the end.

Include all entries on one line with no line breaks. Separate JAR entries with a colon. Do not include any spaces in the list of JAR files. The number of elements is limited only by command-line length (approximately 8 kilobytes on most machines).

The following shows an example:

```
BRM_home/jre/bin/java -Dfile.encoding=utf-8 -cp "BRM_home/apps/cmt/cmt.jar:BRM_home/jars/
xercesImpl.jar:BRM_home/jars/xmlParserAPIs.jar:BRM_home/jars/pcm.jar:BRM_home/jars/
pcmext.jar:BRM_home/apps/cmt" com.portal.cmt.Cmt $1 $2 $3 $4 $5 $6 $7 $8 $9
```

Defining Timestamp Validation for Finite Partitioned Classes

When you import data that is in finite partitioned classes, you can define the timestamp validation mode that is encoded in the POID of a finite partitioned class. You provide the timestamp validation mode as the value for the **infranet.cmt.timestampvalidation** entry in the **Infranet.properties** file for use by Conversion Manager.

Table 2-1 lists the valid entries for **infranet.cmt.timestampvalidation**.

Table 2-1 Valid Modes for Timestamp Validation

Input Value	Timestamp Validation Mode
0	No timestamp validations are performed. If an entry is provided in **created_t**, that creation time is used for a finite partitioned class. Otherwise, the current system time is used for a finite partitioned class.
1	The **created_t** entry is mandatory for any finite partitioned class. It should be provided in the input XML file containing the legacy data to be imported. If **created_t** is not provided, Conversion Manager generates an error. This is the default value for **Infranet.cmt.timestampvalidation**.
2	The default time is used for the POID encoding. If **created_t** is provided for any finite partitioned class, Conversion Manager generates an error.

To add the **infranet.cmt.timestampvalidation** entry to the **Infranet.properties** file:

1. Open the *BRM_home*/**apps**/**cmt**/**Infranet.properties** file in a text editor.
2. Add the following entry. Provide the required value for *n*. See Table 2-1.

   ```
   infranet.cmt.timestampvalidation = n
   ```

> **Note**
>
> If you provide an invalid input for **infranet.cmt.timestampvalidation**, BRM replaces that value with the default value (**1**).

3. Save and close the **Infranet.properties** file.

Viewing Data before Deploying

When you import data, the data is migrated to your production database, but it is not accessible to normal BRM processes until it is deployed. To view the data, you need to configure one or more Oracle Data Manager (DMs) as scoped DMs. A *scoped DM* can view accounts according to their stage ID. You can configure multiple scoped DMs, one for each staging area.

In addition to configuring scoped DMs, you also need to configure the production Oracle DM to support DM scoping.

A BRM system that includes scoped DMs is called a *scoped system*.

To configure a scoped system:

1. Run the **pin_convert_nonunique.sql** script in *BRM_home***/apps/cmt/scripts**. This script drops unique constraints for indexes used in account creation.

   > **Note**
   >
   > By default, the **pin_convert_nonunique.pl** script uses PINX00 to identify tablespaces for indexes. If you use a name other than PINX00, edit the **pin_convert_nonunique.pl** script to change PINX00 to your tablespace name.

2. Run the *BRM_home***/sys/dd/data/cmt_copy_group_planlists_oracle.sql** stored procedure. This procedure duplicates the **/group/plan_list** object with the staged database number, which enables you to create accounts by using Billing Care.

   The stored procedure uses the following parameters:

   - Database number; for example, 5
   - Return code (number)
   - Return message (varchar2(100))

   The following shows a sample call using PL/SQL:

   ```
   declare
   p_return_code number;
   p_return_mesg varchar2(100);
   begin
   CMT_COPY_GROUP_PLANLISTS_ORACLE(5,p_return_code,p_return_mesg);
   end;
   ```

3. Create a **pin.conf** file for each scoped Oracle DM. To do so, edit the Oracle DM **pin.conf** file in *BRM_home***/sys/dm_oracle**.

   Edit the following entries:

   - Set the **scoped_system** entry to **1**. This enables database scoping.
   - Set the **production_system** entry to **0**. This identifies the DM as a scoped DM.

- Set the **staging_db_number** entry to the stage ID used for importing data. If you use a multischema system, the stage ID must be larger than the largest schema database number.
- Include the table names for the data you are importing in the **scoped_exception_tables** entry.

For example, a scoped DM for stage ID 11 has the following entries:

```
- dm scoped_system 1
- dm production_system 0
- dm staging_db_number 11
- dm scoped_exception_tables channel_event_t channel_t config_t data_t deal_t plan_t
product_t rate_plan_selector_t rate_plan_t rate_t search_t zonemap_t
```

4. Repeat the DM **pin.conf** configuration for each scoped DM, using a different stage ID for each one.
5. Configure the production Oracle DM by editing the following entries:
   - Set the **scoped_system** entry to **1**.
   - Set the **production_system** entry to **1**. This identifies the DM as the production DM.
   - Include the table names for the data you are importing in the **scoped_exception_tables** entry.

   The configuration for the production system looks like this:

   ```
   - dm scoped_system 1
   - dm production_system 1
   - dm scoped_exception_tables channel_event_t channel_t config_t data_t deal_t plan_t
   product_t rate_plan_selector_t rate_plan_t rate_t search_t zonemap_t
   ```

6. Start the scoped DMs.
7. Create a root account for each stage ID:
   a. Start the scoped Oracle DMs.
   b. Open the **load_pin_acct** script in *BRM_home*/**apps/cmt/scripts**.
   c. Edit the first line of the script to point to a valid Perl path. For example:

      ```
      /opt/portal/ThirdParty/perl/5.8.0/bin
      ```

   d. Edit the **$dm_port** entry to point to the Oracle DM port number. For example:

      ```
      $dm_port = 22251
      ```

   e. Make sure the **$db_no** entry points to the production database. For example, if the production database number is 0.0.0.1, and the staged database number is 0.0.0.11, use the following entry:

      ```
      $db_no = 0.0.0.1
      ```

   f. Save and close the file.
   g. Run the **load_pin_acct** script using the following command:

      ```
      load_pin_acct -I pin_init_acct
      ```

> **Note**
>
> The input flist used when the root account is created still refers to the production database number because the staging DM translates it to the staging database number.

8. Stop and restart all Connection Managers and Oracle DMs used for accessing the staged data.

Enabling Multischema Loading

If you are migrating data to a multischema system, edit the entries shown in Table 2-2 in the *BRM_home*/**apps/cmt/Infranet.properties** file:

Table 2-2 Infranet.properties File Entries to Enable Multischema Loading

Entry	Description
infranet.cmt.deploy.multidb	Enables or disables data migration in a multischema system. Enter **true** or **false**.
infranet.cmt.primarydbname	Specifies the database schema name (for example, **pindb1**).
infranet.cmt.primarydbuserid	Specifies the schema login name.
infranet.cmt.primarydbpasswd	Specifies the schema login password.
infranet.cmt.primarydbnumber	Specifies the database number for the primary schema (for example, **0.0.0.2**).

Enabling XML Validation

The **pin_cmt** utility can validate the input files to ensure that the data has the correct structure. However, this decreases performance. If you validate the XML prior to loading the data, you can leave validation disabled.

> **Note**
>
> You can test the data mapping by enabling validation when you migrate data to a test database. You can also test the data mapping by using the Conversion Manager XSD file with an online XML validator or third-party tool.

To enable or disable XML validation, edit the **infranet.cmt.preprocess.validation** entry in the **Infranet.properties** file:

```
infranet.cmt.preprocess.validation = true
```

Configuring the Database Setup

Edit the following entries in the **Infranet.properties** file to match your typical account configuration:

- Use the **infranet.cmt.avgnoofservices** entry to specify the average number of services that each account owns. This entry reserves Portal object IDs (POIDs) for the objects that will be created by the **pin_cmt** utility. The default is **5**.
- Use the **infranet.cmt.avgnoofdevices** entry to specify the average number of devices that each account owns. This entry reserves POIDs for the objects that will be created by the **pin_cmt** utility. The default is **2**.
- Use the **infranet.cmt.noofrecords** entry to specify the average number of account records in each input XML data file.
- Use the **infranet.cmt.dbnumber** entry to set the staged database number (for example, **0.0.0.12**). Use the **infranet.cmt.dbname**, **infranet.cmt.userid**, and **infranet.cmt.passwd** entries for the rest of the staged database information.
- Use the **infranet.cmt.targetdbnumber** entry to set the target database.

> **Note**
>
> If an account owns more or fewer services or devices than the numbers you specify, the objects are still loaded.

Setting the Default Credit Limit Profile

When accounts are imported, balance groups are created for the accounts.

You can specify which credit limit profile is assigned to each balance group in the following ways:

- You can specify the default credit limit profile by editing the **infranet.cmt.creditprofile** entry in the **Infranet.properties** file:

  ```
  infranet.cmt.creditprofile = IndexID
  ```

  Each credit limit profile is stored in a PIN_FLD_PROFILES array in the **/config/credit_profile** object. Replace *IndexID* with the index ID of the appropriate PIN_FLD_PROFILES array. In this example, you would replace *IndexID* with **3**:

  ```
  0 PIN_FLD_PROFILES                ARRAY [3] allocated 3, used 3
  1    PIN_FLD_CREDIT_FLOOR         DECIMAL [0] NULL
  1    PIN_FLD_CREDIT_LIMIT         DECIMAL [0] 133
  1    PIN_FLD_CREDIT_THRESHOLDS    INT [0] 0
  ```

- You can assign a credit limit, credit floor, and credit threshold to each balance group when you run the **pin_cmt** utility. To do so, add the following XML elements to the input XML file that you import with **pin_cmt**:
  - **<CrdLmt>**
  - **<CrdFlr>**
  - **<CrdTrsh>**

  The values specified in the input XML file take higher precedence than the value of the **infranet.cmt.creditprofile** entry in the **Infranet.properties** file.

Applying Cycle Fees to Deployed Accounts

By default, the **pin_cmt** utility runs a separate utility, **cmt_mta_cycle_fees**, to apply cycle fees after deploying accounts. To disable this, edit the **infranet.cmt.deploy.opcode** entry in the *BRM_home***/apps/cmt/Infranet.properties** file:

```
infranet.cmt.deploy.opcode = false
```

Migrating New Balances to an Account Without Deleting Existing Balances

By default, when you migrate new balances to an account that was previously migrated using Conversion Manager, the **pin_cmt** utility deletes the existing balances on the account and associates only the newly migrated balances to the account.

To migrate new balances to an account without deleting the existing balances:

1. Open the *BRM_home***/apps/cmt/Infranet.properties** file in a text editor.
2. Add the following entry:

   ```
   infranet.cmt.deleteexistingbalances = false
   ```

3. Save and close the file.

Supporting 31-Day Billing

If your BRM implementation uses 31-day billing, configure the following entries in the **Infranet.properties** file:

```
infranet.cmt.31daybilling = true
infranet.cmt.31daybilling.forward = true
```

If the **infranet.cmt.31daybilling.forward** entry is enabled, BRM performs forward billing if the billing day of month is greater than 28. If it is disabled, BRM performs backward billing.

For more information, see "About Using 31-Day Billing" in *BRM Configuring and Running Billing*.

Supporting Delayed Billing

If your BRM implementation uses delayed billing, configure the following entry in the **Infranet.properties** file:

```
infranet.cmt.billingdelay=true
```

For more information, see "Setting Up Delayed Billing" in *BRM Configuring and Running Billing*.

Improving Conversion Manager Performance

Conversion Manager performance depends on the data and features used for a given BRM implementation. Therefore, this documentation includes guidelines, but does not provide specific values for performance-related configuration settings.

XML File Formatting

Limit the number of accounts in an XML file to 10,000 accounts. Conversion Manager supports XML files with any number of accounts, but limiting the number of accounts enables you to debug the migration in the case of an error message.

When you create the XML files:

- Validate the files with the Conversion Manager physical XML schema.
- Use the proper tab (\t) and new line (\n).
- Do not use spaces.
- Make sure all the required tags are present in the XML.

Running Multiple Instances of the pin_cmt Utility

Use multiple instances of the **pin_cmt** utility to load multiple XML files simultaneously.

For example, you can write a batch/shell script similar to this:

```
cat >> run.sh << EOF
date > start.txt
pin_cmt -import -file one.xml 1 &
pin_cmt -import -file two.xml 1 &
pin_cmt -import -file three.xml 1 &
pin_cmt -import -file four.xml 1 &
pin_cmt -import -file five.xml 1 &
wait
date > end.txt
EOF
```

While running the script, measure the CPU load and time taken by the **pin_cmt** utility for this entire batch. You can then increase the number of **pin_cmt** instances until the CPU load reaches 90%.

Using Connection Pooling with Conversion Manager

To improve performance, you can use connection pooling with Conversion Manager. Configure the following entries in the **Infranet.properties** file:

- **infranet.cmt.minconns**
- **infranet.cmt.maxconns**
- **infranet.cmt.timeout**

For example:

```
infranet.cmt.minconns = 15
infranet.cmt.maxconns = 30
infranet.cmt.timeout = 1000
```

For more information, see "About Connection Pooling" in *BRM System Administrator's Guide*.

Configuring Log File Levels

For test migrations, the default log file settings are set to return troubleshooting information. For better performance, you should change those settings to report less information.

- In the *BRM_home*/**apps/cmt/pin.conf** file, change the **pin_mta loglevel** entry to **1**.
- In the *BRM_home*/**apps/cmt/Infranet.properties** file, change the **infranet.log.level** entry to **1**.

System Resources

Migrating data can use a lot of system resources. Performance can depend on:

- RAM size
- Number of CPUs
- Disk quotas
- File system usage

Conversion Manager Preload Tuning

After you have validated the XML files and are ready to load data, edit the *BRM_home*/**apps/cmt/Infranet.properties** file to maximize performance.

- Turn off XML validation:

  ```
  infranet.cmt.preprocess.validation = false
  ```

- Set the following entries to correspond to the amount of data in the XML files. The **infranet.cmt.noofrecords** entry specifies the number of accounts.

  ```
  infranet.cmt.noofrecords      = 1000
  infranet.cmt.avgnoofservices  = 5
  infranet.cmt.avgnoofdevices   = 2
  ```

- Configure sufficient threads in the preprocess thread pool.

  ```
  infranet.cmt.preprocess.num_of_threads = 5
  ```

  The default value is **5**, which is the optimal value in most cases.

- Make sure there are enough connections in the staging database connection pool:

  ```
  infranet.cmt.minconns   = 15
  infranet.cmt.maxconns   = 30
  infranet.cmt.timeout    = 1000
  ```

Increasing Memory Allocation to Prevent a System Hang

To test your mapping, you can run the **pin_cmt** utility with the default memory allocation. However, when migrating a production database, you might need to increase the memory allocation to prevent a system hang.

Edit the **pin_cmt** script to avoid system hangs.

> **Note**
>
> Depending on your system configuration and load, your settings might be different.

```
JAVA_OPTIONS="-Xms1024m -Xmx2048m"
BRM_home/jre/bin/java $JAVA_OPTIONS -Dfile.encoding=utf-8 -cp "BRM_home/apps/cmt/
cmt.jar:BRM_home/jars/xercesImpl.jar:BRM_home/jars/xmlParserAPIs.jar:BRM_home/jars/
```

```
pcm.jar:BRM_home/jars/pcmext.jar:BRM_home/apps/cmt"
com.portal.cmt.Cmt $1 $2 $3 $4 $5 $6 $7 $8 $9
```

Conversion Manager Load Tuning

In the *BRM_home*/**apps/cmt/Infranet.properties** file, verify the command line parameters for the SQL*Loader.

```
infranet.cmt.sqlldr = sqlldr streamsize=2621440 readsize=2621440 columnarrayrows=5000
```

When you use the flag **direct="true"**, make sure the BRM user has **DBMS_LOCK** permission.

Conversion Manager Deploy Tuning

Before you deploy data, edit the following entries in the Conversion Manager pin.conf file. The file is located in *BRM_home*/**apps/cmt.**

- Set the log level to 1:

  ```
  - pin_mta loglevel 1
  ```

- Specify the appropriate capacity settings for your system; for example:

  ```
  - pin_mta   children     5
  - pin_mta   per_batch    500
  - pin_mta   per_step     1000
  - pin_mta   fetch_size   5000
  ```

3

Mapping Legacy Data to the BRM Data Schema

This chapter explains, in general terms, what you need to understand about your legacy data to successfully convert it to an Oracle Communications Billing and Revenue Management (BRM) database.

Topics in this document:

See also "Understanding Conversion Manager".

About Creating XML Files

To migrate data, you create XML files that contain the data. You then migrate the data to the BRM database by using the **pin_cmt** utility.

> **Note**
>
> Ensure that there are no extra spaces in the input XML file. If you need to indent text in the XML file, use the TAB key to add space.

Sample XML files for each type of data are available in *BRM_home***/apps/cmt/sample_data**. *BRM_home* is the directory in which the BRM server software is installed.

About the XSD Files

Conversion Manager includes two sets of XSD files:

- A set of *conceptual* files that you can use for mapping legacy data to BRM objects. These files are easier to read than the set of files used for converting data.
- A set of *physical* files that are used by Conversion Manager when converting data. These files are not as easy to read, but are written in a way that optimizes performance.

The difference between the types of files is how the XML tags are named. In the conceptual files, the XML tags use the same field names as the BRM object fields.

For example:

- In the conceptual files:

  PIN_FLD_ACCOUNT_NO = <FldAccountNo>

- In the physical files:

  PIN_FLD_ACCOUNT_NO = <ActNo>

To set up mapping, you can use the easily-readable conceptual files to determine how to map data. Then you can edit the physical files, after you know how the data is mapped.

The conceptual files are in *BRM_home***/apps/cmt/schema_files/conceptual**.

The physical files are in *BRM_home***/apps/cmt/schema_files/physical**.

Use the following physical XSD files to create your XML files:

- Use the **CMT_Subscriber.xsd** file for subscriber data.
- Use the **CMT_Balances.xsd** file for balance data.

> **Note**
>
> – Before you use the **pin_cmt** utility to migrate data, validate the XML files with the physical XSD files.
>
> – Do not use spaces while generating the XML files. Use tabs (**\t**) and new lines (**\n**).

About the Types of Data to Convert

Table 3-1 shows the types of legacy data that can be loaded into the BRM database.

Table 3-1 Legacy Data Storable in BRM Database

Type of Data	Description
Account data	Subscriber data such as name, address, profile, current account balances, charge sharing group, and account hierarchy.
Charge offers, discount offers, and bundles	Bundles purchased by the customer and associated with the account.
Bill	Bill information such as billing cycle and balances from the legacy billing system.
Services	Data pertaining to services owned by the account, such as a wireless data service. For example, when converting data for a wireless data service, the following data is also converted: • Device SIM • Device number

Tables Affected by the Conversion Process

Table 3-2 shows the BRM tables that are affected by the conversion process. For more information on the BRM tables and the fields in each table, see the following documents:

- "Storable Class-to-SQL Mapping" in *BRM Developer's Reference*
- "Storable Class Definitions" in *BRM Developer's Reference*

> **Note**
>
> The database tables are created during BRM installation.

Table 3-2 Tables Affected by the Conversion Process

Table Name	Description
ACCOUNT_T	Primary account table that represents billable accounts in BRM. Only one row is added to this table for each account in BRM.
ACCOUNT_EXEMPTIONS_T	Contains an entry for each tax exemption that applies to an account.
ACCOUNT_NAMEINFO_T	Contains name and address information for accounts. There is one row for each name and address type (home address, work address, mailing address, and so forth). There must be at least one row in this table for each row in ACCOUNT_T. For example, you can have billing name and address information, technical contact name and address information, and sales name and address information.
ACCOUNT_PHONES_T	Contains a phone number and a phone type for each account. There is one row for each phone type (up to seven types are defined, including home, work, fax, and pager). If there is no phone number information for an account, there are no rows in the table for the account.
BALANCE_GROUP_T	Stores the balance information such as dollars, minutes, bytes, and frequent flier miles for various balance groups in an account. A balance group includes one or more sub-balances for each balance element. The sub-balance includes its current amount, validity dates, rollover data, and contributors.
BAL_GRP_BALS_T	Stores balance group data.
BAL_GRP_SUB_BALS_T	Stores sub-balance data.
BILL_T	Includes billing information, such as the amount due, amount adjusted, currency, and bill number. A **/bill** object is created at the end of a billing cycle. A **/bill** object is created for every account. The account receivable for a bill is stored in the **/item** objects that point to the balance groups associated with the bill. The **/bill** object points to the **/account** object, the account's bill unit (**/billinfo** object), and the **/invoice** object.
BILLINFO_T	Stores all billing, payment method, accounting cycle, and hierarchy information necessary to bill an account. A bill unit is created for every account. If the bill unit is nonpaying, the **/billinfo** object points to the paying parent **/billinfo** object.
DEVICE_T	Stores information about devices. There is a separate **/device** object for every device managed by BRM. Generic data applicable to all devices is stored in the parent **/device** object. Subclasses such as **/device/num** store information specific to a particular device type.
DEVICE_NUM_T	Stores device information specific to phone numbers managed by Number Manager.
DEVICE_SERVICES_T	Stores device/service mapping data.

Table 3-2 (Cont.) Tables Affected by the Conversion Process

Table Name	Description
DEVICE_SIM_T	Stores device information specific to SIM cards managed by SIM Manager.
EVENT_T	Contains a row for each account that has a nonzero balance forward amount. This table stores data for every event that occurs for an account or service. In the case of conversion of an account that has a balance, you should add a row to reflect the posting of that balance. You also need to write an entry to this table for each entry you write to the EVENT_BILLING_DEAL_INFO_T and EVENT_BILLING_PRODUCT_ACTION_T tables. In addition, you can optionally add rows to this table to store past activity for an account. For example, use this table to add rows for importing past billing and payment history into BRM. These rows can be brought over as memo type events, which can be viewed, but they do not affect the current account balance (which should be calculated and posted separately).
EVENT_BAL_IMPACTS_T	Stores event data.
EVENT_ESSENTIALS_T	Stores event data.
GROUP_T	Represents collections of other objects that have an assigned set of shared attributes. This table is optional. If you use it at conversion time, one row must be added to this table if the account is a parent account.
GROUP_BILLING_MEMBERS_T	Rows for this table are optional and are included only for child accounts (that have a parent account). During conversion, one row needs to be added to this table for each child account (regardless of the child account's payment method).
GROUP_PERMITTEDS_T	Indicates which accounts are group accounts. Rows for this table are optional and are included only for parent accounts that have at least one child account.
GROUP_SHARING_MEMBERS_T	Stores members of a sharing group.
GROUP_SHARING_CHARGES_T	Stores charges of a sharing group.
GROUP_SHARING_DISCOUNTS_T	Stores discounts of a charge sharing group.
GROUP_SHARING_PROFILES_T	Stores the profile data that is shared in a profile sharing group.
ITEM_T	Created to bundle events, this table summarizes billable item activity by type. Rows in this table are added for each row in the BILL_T table. In a conversion scenario, only one type of ITEM_T entry is important, the **/usage** item. Although there are others (cycle forward item, payment item, dispute item, and so on), they are normally not pertinent for conversion. For conversion, a default **/item/misc** is created to bundle events. Also, a service item is created during conversion only if the recreate **='/item/<item-type>'** is specified in the input XML data.
JOURNAL_T	Stores general ledger (G/L) journal data.
ORDERED_GROUPS_T	Stores ordered balance group data.
ORDERED_BALGROUP_T	Stores ordered balance group data.

Table 3-2 (Cont.) Tables Affected by the Conversion Process

Table Name	Description
PAYINFO_T	Stores generic payment method information for an account. Only one row is added in this table for each row in ACCOUNT_T. In addition to a row in PAYINFO_T, there must also be a row added to the applicable payment method table. For example, you have to add a row to PAYINFO_INV_T if the payment method for an account is **Invoice**. Additionally, if you have created a custom payment method and a PAYINFO_*paymentMethod*_T table, where *paymentMethod* is the custom payment method, you must add a row to that table for each account with that payment method.
PAYINFO_CC_T	Contains a row for each account that has a payment method of **Credit Card**. This row is in addition to a row in PAYINFO_T. **Note:** This payment method assumes that you are using the BRM-provided FirstUSAPaymentech or FDC modules to process your credit cards or that the clearing house or bank you use can process the same information. If your credit card processing is significantly different, you might want to set it up as another payment method and store information on that payment method separately. If you migrate tokenized credit or debit card numbers, ensure that you add the card type value for each credit card account. This ensures that the legacy credit and debit card data is migrated in the same format used for storing the credit card data in the PAYINFO_CC_T table. The following card type values are used in the PAYINFO_CC_T table: • **1** for VISA card • **2** for MASTER card • **3** for American Express card • **5** for Discover card • **6** for Diners Club card • **7** for Carte Blanche • **8** for JCB • **9** for SWITCH • **10** for unknown card types
PAYINFO_DD_T	Contains a row for each account that has a payment method of **Direct Debit**. This row is in addition to a row in PAYINFO_T.
PAYINFO_INV_T	Contains a row for each account that has a payment method of **Invoice**. Optionally, the table can also contain purchase order (PO) information.
PROFILE_T	Contains profile information, if any. This table can be left empty if you decide not to convert profile information. You can populate only one row in this table for each row in ACCOUNT_T.
PROFILE_customTable_T	Rows are added to this table only if rows are added to PROFILE_T, where *customTable* is the unique identifier you choose; for example, company name. For each row added to PROFILE_T, a corresponding row is added to PROFILE_**customTable**_T.
PROFILE_ACCT_EXTRATING_DATA_T	Contains profile information.
PROFILE_SERV_EXTRATING_T	Contains profile information.

Table 3-2 (Cont.) Tables Affected by the Conversion Process

Table Name	Description
PROFILE_SERV_EXTRATING_DATA_T	Contains profile information.
PURCHASED_DISCOUNT_T	Contains an entry for each discount owned by an account at the time of conversion.
PURCHASED_PRODUCT_T	Contains an entry for each charge offer owned by an account at the time of conversion.
SERVICE_T	Stores generic service type information for accounts. There is one row in this table for each applicable service for each entry in ACCOUNT_T. In addition to a row in this table, a row must be created in the service type table, such as broadband service or email.
SERVICE_ALIAS_LIST_T	Stores service aliases, used to identify customers by phone number.
SERVICE_TELCO_T	Stores telco service data.
SERVICE_TELCO_FEATURES_T	Stores telco service data.
SERVICE_TELCO_GSM_T	Stores telco service data.
UNIQUENESS_T	Stores data that enforces uniqueness of logins across different database schemas in a multischema environment.

In addition, the following audit tables are affected:

- **AU_ACCOUNT_T**
- **AU_BAL_GRP_T**
- **AU_GROUP_SHARING_CHARGES_T**
- **AU_GROUP_SHARING_DISCOUNTS_T**
- **AU_GROUP_SHARING_PROFILES_T**
- **AU_GROUP_T**
- **AU_ORDERED_BALGROUP_T**
- **AU_ORDERED_GROUPS_T**
- **AU_PROFILE_ACCT_EXTRATING_DATA_T**
- **AU_PROFILE_SERV_EXTRATING_DATA_T**
- **AU_PROFILE_SERV_EXTRATING_T**
- **AU_PROFILE_T**
- **AU_PURCHASED_DISCOUNT_T**
- **AU_PURCHASED_PRODUCT_T**
- **AU_SERVICE_ALIAS_T**
- **AU_SERVICE_T**
- **AU_UNIQUENESS_T**

Loading Data into Additional Audit Tables

You can configure Conversion Manager to load legacy data into audit tables that are not listed in "Tables Affected by the Conversion Process" by:

1. Creating control files for each additional audit table
2. Adding the list of additional audit tables to the **pin_cmt** utility's **Infranet.properties** file

The following procedure describes how to configure Conversion Manager to load legacy data into the **AU_ACCOUNT_NAMEINFO_T** audit table, but you can follow a similar procedure to load legacy data into other audit tables.

To load legacy data into the **AU_ACCOUNT_NAMEINFO_T** audit table:

1. Go to the *BRM_home***/apps/cmt/ctl_files** directory.
2. Copy the control file for the **ACCOUNT_NAMEINFO_T** table to the *BRM_home***/apps/cmt/ctl_files/audit** directory:

   cp account_nameinfo_t.ctl *BRM_home***/apps/cmt/ctl_files/audit/account_nameinfo_t.ctl**
3. Open the *BRM_home***/apps/cmt/ctl_files/audit/account_nameinfo_t.ctl** file in a text editor.
4. Change all instances of ACCOUNT_NAMEINFO_T to AU_ACCOUNT_NAMEINFO_T.

   For example, change these lines:

   ```
   LOAD DATA
   APPEND
   INTO TABLE ACCOUNT_NAMEINFO_T
   ```

   to the following:

   ```
   LOAD DATA
   APPEND
   INTO TABLE AU_ACCOUNT_NAMEINFO_T
   ```
5. Save and close the file.
6. Open the *BRM_home***/apps/cmt/ctl_files/Infranet.properties** file in a text editor.
7. Add **account_nameinfo_t** to the **infranet.cmt.auditloaders** line:

   ```
   infranet.cmt.auditloaders = account_t; bal_grp_t; group_sharing_charges_t;...;
   account_nameinfo_t;
   ```
8. Save and close the file.

4

Migrating Data by Using New and Extended Storable Classes

This chapter describes how to migrate data to your Oracle Communications Billing and Revenue Management (BRM) database when the data requires new or extended storable classes.

Topics in this document:

See also "Understanding Conversion Manager".

About Migrating Data by Using New and Extended Storable Classes

If your legacy data includes data not currently supported in BRM, you support the custom data by creating storable classes or extending storable classes. You can then extend the Conversion Manager XML schema to migrate the data.

> **Note**
>
> Conversion Manager does not support extensions for all BRM storable classes. See "About Extended Storable Classes for Migration".

To use new or extended storable classes:

- Create the new or extended storable classes. See "About Extended Storable Classes for Migration".
- Create XML files that include the migrated data. See "Creating XML Files for New or Extended Storable Class Data".
- Create control files to add tables to the BRM database. See "Creating Control Files for Extended Storable Classes".
- Run the **pin_cmt** utility as you would for migrating nonextended data. See "Loading Legacy Data into the BRM Database".

> **Note**
>
> – When you import data for new storable classes, you use a different **pin_cmt** utility parameter (**-import_custom**). When you migrate data into custom storable classes, the **pin_cmt** utility performs the additional step of activating the custom data when it is deployed.
>
> – Import all custom data before you deploy any data. The only exception is if all the custom data belongs to the root account; for example, custom system configuration data.

About Extended Storable Classes for Migration

You can extend the following storable classes:

- **/service**
- **/device**
- **/payinfo**
- **/profile**

For example:

- **/service/extension1**
- **/service/extension1/extension**

About Linking to Data from New or Extended Storable Class Data

Data in new or extended storable classes can include links to other objects. For example, a **/service/**extended object can include a link to an **/account** object by including the account ID in the PIN_FLD_ACCOUNT_OBJ field.

You can link to the following types of objects:

- **/account**
- **/service** (and extended service storable classes)
- **/device** (and extended device storable classes)
- **/payinfo** (and extended payinfo storable classes)
- **/profile** (and extended profile storable classes)
- **/group/sharing/charges**
- **/group/sharing/discounts**

When creating new or extended storable classes, follow the BRM standards and restrictions. For example:

- You cannot nest substructs.
- An array can include an array, but no further levels are allowed.

Creating XML Files for New or Extended Storable Class Data

Use these guidelines when creating an XML file for data for new or extended storable classes:

- Use the XML **myExtension** element for extended data.
- Use the XML **NewClass** element for new custom storable classes.
- Use the **myArray** and **mySubstruct** elements for arrays and substructs in the extended data.
- Use the **table** attribute in each array and substruct to define the table that stores the data.
- Use the **elem** attribute in each array to define the index.
- Use the **RefObj** element to create links to other objects. To do so, use the ID of the object being linked to. For example:

  ```
  <RefObj type ="/account"> 1234 </RefObj>
  ```

  If the **pin_cmt** utility does not find the object being linked to, it reports a warning and inserts a null value in the Portal object ID (POID) being referenced.
- Use the **type** attribute in all new and extended elements to define the type of data.
  - The **type** attribute for a *new* storable class is defined in the base storable class element. For example:

    ```
    <NewClass  id="1243" type="/myclass" table="myclass_t" read="L" write="L" >
    ```
  - The **type** attribute for an *extended* storable class is defined in the base storable class element. For example:

    ```
    <ActSrv id="771" type="/service/extended" precreate="/item/misc" global="true">
    ```

For XML examples, see the following sections:

- Example of Extending a Service Storable Class
- Example of Extending a Device Storable Class
- Example of Creating a Storable Class

Creating an XSD File for Extended Data

To validate the XML file, you must create an XSD file for the extended information. You can use the **myExtension.xsd** file in *BRM_home***/apps/cmt/sample_data** as a starting point. *BRM_home* is the directory in which the BRM server software is installed.

> **Note**
>
> For each type of extended information, use a different namespace to avoid collisions caused by using the same name **myExtension**.

If the extended information contains BRM extensions (for example, **/service/***x*, **/payinfo/***x*, **/device/***x*, and **/profile/***x*), include the custom XSD in the **CMT_Subscribers.xsd** file. This allows the extended information to be validated along with the subscriber information.

If the extended information includes a custom storable class (for example, **/my_profile**), the custom XSD does not need to be included in the **CMT_Subscribers.xsd** file. The extended information can be validated by the custom XSD file.

Put the custom XSD file in the folder specified in the **infranet.cmt.schema.location** entry in the **pin_cmt** utility **Infranet.properties** file. By default, the folder is *BRM_home***/apps/cmt/schema_files/physical**.

> **Note**
>
> To enable XML validation, use the **infranet.cmt.preprocess.validation** entry in the **pin_cmt Infranet.properties** file. See "Enabling XML Validation".

Creating Control Files for Extended Storable Classes

Control files add tables to the BRM database. You create a control file for each table. You need to create a control file for each of the following:

- New storable classes
- Substructs in new or extended storable classes
- Arrays in new or extended storable classes

When you create a control file, the name of the control file is used as the table name. For example, if you create a new storable class named **/myclass**, which includes a substruct named PIN_FIELD_MYSUBSTRUCT, the table name would be MYCLASS_SUBSTR_T. The control file name would be **myclass_substr_t.ctl**.

> **Note**
>
> - The order of columns in the control file must be the same as the order of columns in the table.
> - The order of data in the XML input file must be the same as the order of data in the control file and table.
> - While adding tables to the BRM database, ensure that the tables listed in the **pin_cmt** utility **Infranet.properties** file belong to the main storable class.

For control file examples, see the following sections:

- Example of Extending a Service Storable Class
- Example of Extending a Device Storable Class
- Example of Creating a Storable Class

To use your custom control files:

1. Put the control files in *BRM_home***/apps/cmt/ctl_files**.
2. Edit the **pin_cmt** utility **Infranet.properties** file.
   - For control files that add substruct and array data, edit the **infranet.cmt.loaders** entry. Add the control file names. For example:

     ```
     infranet.cmt.loaders = account_t; account_nameinfo_t; ...; extended2_substr1_t;
     extended_cust1_t; extended2_cycle_t;
     ```

   - For control files that add data for a new storable class, add the **infranet.cmt.customtables** entry. Add the table names, separated by semicolons. For example:

     ```
     infranet.cmt.customtables = myclass_t;
     ```

> **Note**
>
> By default, the **infranet.cmt.customtables** entry is not included in the **Infranet.properties** file.

Creating Control Files for Custom Event Tables in a Virtual Column-Enabled System

BRM Conversion Manager includes control files that are suitable for a virtual column-enabled system for migrating event tables (the **event_t** and **event_bal_impacts_t** tables). If you have custom event tables to be migrated (for event-type objects that are not in the BRM system), you must create the control files as described in this chapter. In addition, in a virtual column-enabled system, the BRM Conversion Manager event control files cannot perform insert operations on the virtual columns (the *field_name*_**type** columns) of event tables in the BRM database; they must, instead, perform insert operations on the virtual column's associated supporting column *field_name*_**type_id**.

> **Note**
>
> BRM's support for virtual columns is deprecated in BRM 15.2 and will be removed in a future release.

See *BRM System Administrator's Guide* for information about using virtual columns on event tables.

Example of Extending a Service Storable Class

The following example shows the XML data for an extended **/service** storable class:

```
<ActSub>
    <ActSrv id="771" type="/service/extended" precreate="/item/misc" global="true">
        <CWhn>2004-10-02T01:22:15-07:00</CWhn>
        <Log>159-51090219-237205-0-24083-1-tahoe</Log>
        <Pass>md5|5f4dcc3b5aa765d61d8327deb882cf99</Pass>
        <SrvSta>A</SrvSta>
        <Eff>2004-02-02T01:22:15-07:00</Eff>

        <SClsInfo>
        <myExtension>
            <mySubstruct table="extended_substr1_t" >
                <FldCustName type="string">abc</FldCustName>
                <FldCustType type="integer">99</FldCustType>
            </mySubstruct>
            <myArray table="extended_cust1_t" elem="1">
                    <FldAmt type="integer">190</FldAmt>
                    <FldDate="date">2004-02-02T01:22:15-7:00</FldDate>
                    <myArray table="extended_cycle_t" elem="1">
                        <Fldtype type="integer">190</Fldtype>
                    </myArray>
                </myArray>
        </myExtension>

        </SClsInfo>
```

```
    </ActSrv>
</ActSub>
```

In Developer Workshop, the extended **/service** storable class definition looks like this:

```
/service/extended
    CMT_ARRAY
        CMT_NAME
        PIN_FLD_ACCOUNT_TYPE
    CUST_ARRAY
        PIN_FLD_CYCLE
            PIN_FLD_CUSTOMER_TYPE
        PIN_FLD_CYCLE_DISC_AMT
        PIN_FLD_CYCLE_END_T
```

After loading the data, the extended fields are included in the stored object like this:

```
0 PIN_FLD_POID                    POID [0] 0.0.0.1 /service/extended 11813 3
0 PIN_FLD_CREATED_T               TSTAMP [0] (1083337746) Fri Apr 30 08:09:06 2004
0 PIN_FLD_MOD_T                   TSTAMP [0] (1083338129) Fri Apr 30 08:15:29 2004
.
.
.
0 PIN_FLD_TYPE                    ENUM [0] 0
0 PIN_FLD_ALIAS_LIST              ARRAY [0] allocated 1, used 1
1     PIN_FLD_NAME                STR [0] "00491110102"
0 10000                           SUBSTRUCT [0] allocated 2, used 2
1     10004                       STR [0] "abc"
1     PIN_FLD_ACCOUNT_TYPE        ENUM [0] 99
0 10005                           ARRAY [1] allocated 3, used 3
1     PIN_FLD_CYCLE_DISC_AMT      DECIMAL [0] 190
1     PIN_FLD_CYCLE_END_T         TSTAMP [0] (1075713735) Mon Feb  2 01:22:15 2004
1     PIN_FLD_CYCLE               ARRAY [1] allocated 1, used 1
2         PIN_FLD_CUSTOMER_TYPE   ENUM [0] 190
```

The following examples show the control files used for loading the data for the extended **/service** storable class. The substruct and arrays require new tables. The array tables include the columns REC_ID or REC_ID2; the substruct does not include these columns.

Example Control File for Substruct

```
-- extended_substr1_t.ctl

LOAD DATA
APPEND
    INTO TABLE EXTENDED_SUBSTR1_T
    (
        OBJ_ID0    INTEGER EXTERNAL TERMINATED BY ',',
        NAME       CHAR TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"',
        TYPE       INTEGER EXTERNAL TERMINATED BY ','
     )
```

Example Control File for Array

```
-- extended_cust1_t.ctl

LOAD DATA
APPEND
    INTO TABLE EXTENDED_CUST1_T
    (
        OBJ_ID0     INTEGER EXTERNAL TERMINATED BY ',',
        REC_ID      INTEGER EXTERNAL TERMINATED BY ',',
        AMOUNT      INTEGER EXTERNAL TERMINATED BY ',',
```

```
        END_T       INTEGER EXTERNAL TERMINATED BY ','
     )
```

Example Control File for Array

```
-- extended_cycle_t.ctl

LOAD DATA
APPEND
    INTO TABLE EXTENDED_CYCLE_T
    (
        OBJ_ID0    INTEGER EXTERNAL TERMINATED BY ',',
        REC_ID     INTEGER EXTERNAL TERMINATED BY ',',
        REC_ID2    INTEGER EXTERNAL TERMINATED BY ',',
        TYPE       INTEGER EXTERNAL TERMINATED BY ','
    )
```

Setting a Service Balance Group

The following example shows attributes in the input XML file that you set to associate a balance group with a service.

To create a balance group and associate it with a service, set the **bal_grp** attribute to **true**.

```
<ActSrv id="50001" type="/service/telco/gsm/telephony" precreate="/item/misc"
global="true" bal_grp="true" billInfoElem="1">
        <Log>159-20040202-210000-0-24083-1-zion</Log>
        <Pass>md5|5f4dcc3b5aa765d61d8327deb882cf99</Pass>
        <SrvSta>A</SrvSta>
        <CrtT>2004-05-26T01:22:15-07:00</CrtT>
        <Eff>2004-05-26T01:22:15-07:00</Eff>
        <ServId>ServiceTelephony</ServId>
        </ActSrv>
```

To associate the balance group of a bill unit (**/billinfo** object) with a service, set the **skip_bal_grp_crt** and **bal_grp** attributes to **true**. The balance group of the bill unit is associated with the service.

```
<ActSrv id="50001" type="/service/telco/gsm/telephony" precreate="/item/misc"
global="true" bal_grp="true" skip_bal_grp_crt="true" billInfoElem="1">
        <Log>159-20040202-210000-0-24083-1-zion</Log>
        <Pass>md5|5f4dcc3b5aa765d61d8327deb882cf99</Pass>
        <SrvSta>A</SrvSta>
        <CrtT>2004-05-26T01:22:15-07:00</CrtT>
        <Eff>2004-05-26T01:22:15-07:00</Eff>
        <ServId>ServiceTelephony</ServId>
        </ActSrv>
```

Example of Extending a Device Storable Class

The following example shows the XML data for an extended **/device** storable class:

```
<DeviceInfo>
    <Device id="778" type="/device/extended2">
        <DevId>00982110309</DevId>
        <Src>source2</Src>
        <Mnufr>Airtel2</Mnufr>
        <Mdl>A4IR124</Mdl>
    </Device>
    <myExtension>
        <myArray table="dev_extended2_t" elem="1">
```

```
            <FldAmount type="integer">11</FldAmount>
            <FldDate type="date">2004-03-15T01:22:15-07:00</FldDate>
        </myArray>
    </myExtension>
</DeviceInfo>
```

In Developer Workshop, the extended **/device** storable class definition looks like this:

```
/device/extended2
    PIN_FLD_ENTRIES
        PIN_FLD_ENTRY_AMOUNT
        PIN_FLD_ENTRY_T
```

After loading the data, the extended fields are included in the stored object like this:

```
# number of field entries allocated 14, used 14
0 PIN_FLD_POID               POID [0] 0.0.0.1 /device/extended2 11811 0
0 PIN_FLD_CREATED_T          TSTAMP [0] (1083337746) Fri Apr 30 08:09:06 2004
0 PIN_FLD_MOD_T              TSTAMP [0] (0) <null>
0 PIN_FLD_READ_ACCESS        STR [0] "L"
0 PIN_FLD_WRITE_ACCESS       STR [0] "A"
0 PIN_FLD_ACCOUNT_OBJ        POID [0] 0.0.0.1 /account 1 0
0 PIN_FLD_DESCR              STR [0] ""
0 PIN_FLD_DEVICE_ID          STR [0] "00982110309"
0 PIN_FLD_MANUFACTURER       STR [0] "Airtel2"
0 PIN_FLD_MODEL              STR [0] "A4IR124"
0 PIN_FLD_SOURCE             STR [0] "source2"
0 PIN_FLD_STATE_ID           INT [0] 0
0 PIN_FLD_SERVICES           ARRAY [3] allocated 2, used 2
1     PIN_FLD_ACCOUNT_OBJ    POID [0] 0.0.0.1 /account 11806 0
1     PIN_FLD_SERVICE_OBJ    POID [0] 0.0.0.1 /service/extended2 11813 0
0 PIN_FLD_ENTRIES            ARRAY [1] allocated 2, used 2
1     PIN_FLD_ENTRY_AMOUNT   DECIMAL [0] 11
1     PIN_FLD_ENTRY_T        TSTAMP [0] (1079342535) Mon Mar 15 01:22:15 2004
```

The following example shows the control file used for loading the data for the extended **/device** storable class:

```
LOAD DATA
APPEND
    INTO TABLE DEV_EXTENDED2_t
    (
        OBJ_ID0                 INTEGER EXTERNAL TERMINATED BY ',',
        REC_ID                  INTEGER EXTERNAL TERMINATED BY ',',
        AMOUNT                  INTEGER EXTERNAL TERMINATED BY ',',
        ENTRY_T                 INTEGER EXTERNAL TERMINATED BY ','
    )
```

Example of Creating a Storable Class

The following example shows the XML data for a new storable class named **/myclass**:

```
<NewClass  id="1243" type="/myclass" table="myclass_t" read="L" write="L" >
    <AcctObj>0.0.0.1-indian500</AcctObj>
    <myArray table="myclass_rules_t" elem="0">
        <rum_id type="integer">222</rum_id>
        <descr type="string">sachin</descr>
        <start_t type="date">2006-10-02T01:22:15-07:00</start_t>
        </myArray>
    </myArray>
</NewClass>
```

In Developer Workshop, the **/myclass** definition looks like this:

```
/myclass
    PIN_FLD_ACCOUNT_OBJ
    PIN_FLD_CREATED_T
    PIN_FLD_MOD_T
    PIN_FLD_POID
    PIN_FLD_READ_ACCESS
    PIN_FLD_RULES
        PIN_FLD_RUM_ID
        PIN_FLD_SCENARIO_DESCR
        PIN_FLD_SCHEDULE_DOWNTIME_START
    PIN_FLD_WRITE_ACCESS
```

After loading the data, the stored object looks like this:

```
0 PIN_FLD_POID                          POID [0] 0.0.0.1 /myclass 43567 0
0 PIN_FLD_CREATED_T                     TSTAMP [0] (1083780256) 5/6/04 3:04 AM
0 PIN_FLD_MOD_T                         TSTAMP [0] (1083780256) 5/6/04 3:04 AM
0 PIN_FLD_READ_ACCESS                   STR [0] "L"
0 PIN_FLD_WRITE_ACCESS                  STR [0] "L"
0 PIN_FLD_ACCOUNT_OBJ                   POID [0] 0.0.0.1 /account 1 1
0 PIN_FLD_RULES                         ARRAY [0] allocated 4, used 4
1    PIN_FLD_RUM_ID                     INT [0] 222
1    PIN_FLD_SCENARIO_DESCR             STR [0] "sachin"
1    PIN_FLD_SCHEDULE_DOWNTIME_START    TSTAMP [0] (1159732335) 10/2/06 4:52 AM
```

The following example shows the control file used for loading the data for the new **/myclass** storable class:

```
myclass_t.ctl

LOAD DATA
APPEND
    INTO TABLE MYCLASS_T
    (
        POID_DB             INTEGER EXTERNAL TERMINATED BY ',',
        POID_ID0            INTEGER EXTERNAL TERMINATED BY ',',
        POID_TYPE           CHAR TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"',
        POID_REV            INTEGER EXTERNAL TERMINATED BY ',',
        CREATED_T           INTEGER EXTERNAL TERMINATED BY ',',
        MOD_T               INTEGER EXTERNAL TERMINATED BY ',',
        READ_ACCESS         CHAR TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"',
        WRITE_ACCESS        CHAR TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"',
        ACCOUNT_OBJ_DB      INTEGER EXTERNAL TERMINATED BY ',',
        ACCOUNT_OBJ_ID0     INTEGER EXTERNAL TERMINATED BY ',',
        ACCOUNT_OBJ_TYPE    CHAR TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"',
        ACCOUNT_OBJ_REV     INTEGER EXTERNAL TERMINATED BY ','
    )
```

The following example shows the control file used for loading the data for the **/myclass** storable class PIN_FLD_RULES substruct:

```
myclass_rules_t.ctl

LOAD DATA
APPEND
    INTO TABLE MYCLASS_RULES_T
    (
        OBJ_ID0    INTEGER EXTERNAL TERMINATED BY ',',
        REC_ID     INTEGER EXTERNAL TERMINATED BY ',',
        RUM_ID     INTEGER EXTERNAL TERMINATED BY ',',
        DESCR     CHAR TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"',
```

```
    START_T   INTEGER EXTERNAL TERMINATED BY ','
)
```

Example of Migrating Hierarchical Accounts

The following example shows the various attributes that you need to specify for different accounts at different hierarchy levels in the input XML file while performing the migration:

- **Parent account**: The following attributes should exist for the parent account:

  ```
  <ActSbsc id="0.0.0.1-10001" isParent="Y">
  ```

  Set **id** to the unique string by which the account will be identified. Set **isParent** to **Y**.

- **Child account whose bill units are paying**: The following attributes should exist for the child account whose bill units are paying:

  ```
  <ActSbsc id="0.0.0.1-10002" parenRef="0.0.0.1-10001">
  ```

  Set **id** to the unique string by which the account will be identified. Set **parenRef** to a reference to the parent ID.

- **Child account whose bill units are nonpaying**: The following attributes should exist for the child account whose bill units are nonpaying:

  ```
  <ActSbsc id="0.0.0.1-10003" parenRef="0.0.0.1-10001"  payParenRef="0.0.0.1-10001">
  ```

  Set **id** to the unique string by which the account will be identified. Set **parenRef** to a reference to the parent ID. Set **payParenRef** to a reference to the paying parent ID.

  The **pay_type** (**PTyp**) attribute under **billinfo** should be set to **NPC** as follows:

  ```
  <ABinfo global="true">
              <ACDom>26</ACDom>
              <ANxt>2004-07-26T01:22:15-07:00</ANxt>
              <BlWn>1</BlWn>
              <BlSgmnt>RC</BlSgmnt>
              <PTyp>NPC</PTyp>
  </ABinfo>
  ```

  The **payinfo_obj_type** attribute under **payinfo** should be set to **/payinfo.** Also, comment out the **payinfo** section as follows:

  ```
  <APinfo id="10004" type="/payinfo">
                      <DDom>26</DDom>
                      <RelDue>2004-05-26T01:22:15-07:00</RelDue>
                      <!-- payInvExtn>
                             <Add>340/C Baker Street</Add>
                             <Cty>Houston</Cty>
                             <Cntr>USA</Cntr>
                             <Nm>52345</Nm>
                             <Stt>Texas</Stt>
                             <Zip>56009</Zip>
                      </payInvExtn -->
  </APinfo>
  ```

- **Paying and nonpaying child**: Child accounts can have multiple paying bill units and nonpaying bill units:

  - **Paying bill unit**: The following attributes should exist for each paying bill unit:

    ```
    <ActSbsc id="0.0.0.1-10004" parenRef="0.0.0.1-10001">

    <ActSrv id="0.0.1.1-10224" type="/service/telco/fixedretail" billInfoElem="2">
    ```

```
<ABinfo elem="2" parentElem="1" bal_grp="false" payInfoRefId="0.0.0.1-114">
<APinfo id="0.0.0.1-114" type="/payinfo/invoice">
```

- **Nonpaying bill unit**: The following attributes should exist for each nonpaying bill unit:

```
<ActSbsc id="0.0.0.1-10004" parenRef="0.0.0.1-10001">

<ActSrv id="0.0.1.1-10334" type="/service/telco/prepaid" billInfoElem="1">

<ABinfo elem="1" parentElem="1" payingParenRefId="0.0.0.1-10001">
```

  Set **payingParenRefId** to a reference to the parent account ID that owns the paying bill unit. This creates another **/billinfo** record for the child account. The **ar_billinfo_obj_id0** field of that record stores the ID of the paying bill unit.

Setting an Account Bill Unit

The following example shows attributes in the input XML file that you set to define a **/billinfo** object as the account bill unit.

To define a **/billinfo** object as the account bill unit, set the **isAccBillinfo** attribute in the **<ABInfo> /billinfo** object to **Y**. The balance group associated with the **/billinfo** object then becomes the account default balance group.

> **Note**
>
> There can be only one **<ABinfo> /billinfo** object for which the **isAccBillinfo** attribute is set to **Y**. If there are duplicate **<ABinfo> /billinfo** objects, BRM returns an error and the migration stops.

```
<ABinfo global="true" spnrCnt="1" spnreeCnt="2" elem="1" bal_grp="true"
isAccBillinfo="Y" payInfoRefId="MyAPInfo_1">
          <ACDom>21</ACDom>
       <ActgType>B</ActgType>
       <ANxt>2013-07-01T01:22:15-07:00</ANxt>
       <CrtT>2013-06-07T01:22:15-07:00</CrtT>
       <BlWn>1</BlWn>
       <BlSgmnt>RC</BlSgmnt>
       <PTyp>INV</PTyp>
       <BISta>A</BISta>
       <BillStat>3</BillStat>
       <ExmtFrmColl>A</ExmtFrmColl>
       <BIStaFlag>SBCA</BIStaFlag>
       <BillInfoId>MyBillInfo_1</BillInfoId>
</ABinfo>
```

Example of Migrating Hierarchical Bill Units within an Account

The following example shows hierarchical bill units within an account.

> **Note**
>
> To create the hierarchical bill units in the same account, set the **thisActHierarchy** attribute to **true**. In the following example, the bill units **BI1** and **BI2** are in account A. **BI1** is the parent of **BI2**.

```
<ABinfo global="true" spnrCnt="1" spnreeCnt="2" elem="1" bal_grp="true"
payInfoRefId="MyAPInfo_1">
      <ACDom>26</ACDom>
      <ActgType>B</ActgType>
      <ANxt>2013-07-01T01:22:15-07:00</ANxt>
      <CrtT>2013-06-07T01:22:15-07:00</CrtT>
      <BlWn>1</BlWn>
      <BlSgmnt>RC</BlSgmnt>
      <PTyp>INV</PTyp>
      <BISta>A</BISta>
      <BillStat>3</BillStat>
      <ExmtFrmColl>A</ExmtFrmColl>
      <BIStaFlag>SBCA</BIStaFlag>
      <BillInfoId>MyBillId3</BillInfoId>
    </ABinfo>
<ABinfo global="true" spnrCnt="1" spnreeCnt="2" elem="2" parentElem="1"
thisActHierarchy="true" bal_grp="true" payInfoRefId="MyAPInfo_1">
      <ACDom>26</ACDom>
      <ActgType>B</ActgType>
      <ANxt>2013-07-01T01:22:15-07:00</ANxt>
      <CrtT>2013-06-07T01:22:15-07:00</CrtT>
      <BlWn>1</BlWn>
      <BlSgmnt>RC</BlSgmnt>
      <PTyp>NPC</PTyp>
      <BISta>A</BISta>
      <BillStat>3</BillStat>
      <ExmtFrmColl>A</ExmtFrmColl>
      <BIStaFlag>SBCA</BIStaFlag>
      <BillInfoId>MyBillId1</BillInfoId>
</ABinfo>
<ABinfo global="true" spnrCnt="1" spnreeCnt="2" elem="3" parentElem="1"
thisActHierarchy="true" bal_grp="true" payInfoRefId="MyAPInfo_1">
      <ACDom>26</ACDom>
      <ActgType>B</ActgType>
      <ANxt>2013-07-01T01:22:15-07:00</ANxt>
      <CrtT>2013-06-07T01:22:15-07:00</CrtT>
      <BlWn>1</BlWn>
      <BlSgmnt>RC</BlSgmnt>
      <PTyp>NPC</PTyp>
      <BISta>A</BISta>
      <BillStat>3</BillStat>
      <ExmtFrmColl>A</ExmtFrmColl>
      <BIStaFlag>SBCA</BIStaFlag>
      <BillInfoId>MyBillId2</BillInfoId>
</ABinfo>
```

5

Loading Legacy Data into the BRM Database

This chapter describes how to load legacy data into your Oracle Communications Billing and Revenue Management (BRM) database.

Topics in this document:

- Importing Data
- Deploying Converted Data
- Reloading Data
- Troubleshooting Conversion Manager
- Testing the Imported Data

Before you can load data, you must extract it from the legacy database into XML files. See:

- Understanding Conversion Manager
- Mapping Legacy Data to the BRM Data Schema

For information about performance tuning, see "Improving Conversion Manager Performance".

Importing Data

You import legacy data into the BRM database one file at a time.

> **Note**
>
> To specify a database connection, edit the **pin_cmt utility Infranet.properties** file in *BRM_home***/apps/cmt**. *BRM_home* is the directory in which the BRM server software is installed.

To import legacy data into the BRM database:

1. Go to *BRM_home***/apps/cmt**.
2. Do one of the following:
   - To import data that is not stored in a new storable class, run the **pin_cmt** utility using the following syntax:

     ```
     pin_cmt -import -file XML_input_data_file stage_ID
     ```

     For example:

     ```
     pin_cmt -import -file data.xml 100
     ```

> **Note**
>
> If you have a multischema system, make sure the stage IDs are larger than the database schema IDs. For example, if you have a schema with the number 0.0.0.5, use stage IDs larger than 5.

- To import data that is stored in a new storable class, run the **pin_cmt** utility using the following syntax:

  ```
  pin_cmt -import_custom -file XML_input_data_file stage_ID
  ```

  For example:

  ```
  pin_cmt -import_custom -file data.xml 100
  ```

See "pin_cmt" for more information.

Possible errors:

- On all parser errors, Conversion Manager rejects the whole file. Take corrective action based on the errors logged in **cmt.pinlog** and resubmit the corrected file.
- A record is rejected and not imported if its reference object is not found. For example, in case of importing a child account when the parent account is not found, the child record is rejected. The error is noted in the log file.
- I/O errors, such as the inability to find or open the specified document. The whole file is rejected and an error is logged.
- You run out of database space. Use **pin_cmt** with the **-recover** parameter to recover your data. For more information, see "Reloading Data".

Deploying Converted Data

When you deploy the data, the staging accounts are made available for production by updating the database ID number in the object Portal object IDs (POID).

> **Note**
>
> To specify a database connection, edit the **pin_cmt utility Infranet.properties** file in *BRM_home***/apps/cmt**.

To deploy your converted data, run **pin_cmt** using the following syntax:

```
pin_cmt -deploy DOM stage_ID
```

where:

- DOM is the billing cycle's day of month.
- *stage_ID* is the identity of the staging area.

See "pin_cmt" for more information.

Reloading Data

If **pin_cmt** runs out of space in your BRM database for data rows, the importing process stops. Data that was not imported can be imported after more space is made available in the database.

> **Note**
>
> To specify a database connection, edit the **pin_cmt utility Infranet.properties** file in *BRM_home***/apps/cmt**.

To import data when the utility runs out of database space:

1. Add space to the database.
2. Read the log files in *BRM_home***/apps/cmt** to find the following information:
   - How many records were processed.
   - The batch ID for the import process that did not complete.
3. Edit the control files:
   a. At the beginning of every control file, replace the string LOAD_DATA with CONTINUE_LOAD_DATA.
   b. In each table's control file, specify the number of records to skip for that table by using the INTO TABLE clause. For example:

      ```
      INTO TABLE account_t
      SKIP 756
      ```

      where **756** is the number of previously processed records.
4. Run the **pin_cmt** utility with the **-recovery load** parameter:

   ```
   pin_cmt -recovery load batch_ID
   ```

   See "pin_cmt" for more information.

Troubleshooting Conversion Manager

When Conversion Manager imports legacy data into the BRM database, it creates a log file (**cmt.pinlog**) with a list of errors and warnings depending on the reporting level set for message logging. (See "Setting the Reporting Level for Logging Messages" in *BRM System Administrator's Guide*.)

In addition, the log file lists successfully processed records and failed records.

To find any error messages, read the **cmt.pinlog** file in the *BRM_home***/apps/cmt** directory.

You can also read the Connection Manager and Data Manager log files.

Common pin_cmt Utility Error Messages

Table 5-1 shows common **pin_cmt** error messages.

Table 5-1 Common pin_cmt Messages

Error Message	Description
CMD_LINE_ARG_ERR	Error in the **pin_cmt** utility command line syntax.
MISSING_RESOURCE_ERR	A required configuration entry in the **pin_cmt** utility **Infranet.properties** file is missing.
DB_CONNECTION_ERR	The database connection configuration in the **pin_cmt** utility **Infranet.properties** file is incorrect. This error occurs when the database is down.
BAD_INFRANET_CONNECTION	One of the following: • BRM is not running • The CM connection information in the **pin_cmt** utility **Infranet.properties** file is incorrect.
FILE_NOT_FOUND_ERR	The input XML file is missing from the location specified in the command line.
PARSING_ERR	The input XML file is either not well-formed or not valid with respect to CMT XSD.
IL_PR_PARENT_NOT_FOUND_ERR	The input XML file includes an incorrect parent (**/group/billing**) reference.
IL_PR_PAYING_PARENT_NOT_FOUND_ERR	The input XML file includes an incorrect paying parent (**/group/billing**) reference.
INCORRECT_DEVICE_REF	The input XML file includes an incorrect device reference.
INCORRECT_SUB_OBJ_SERVICE_REF	The input XML file includes an incorrect subscription service reference.
INCORRECT_GSC_PARENT_REF	The input XML file includes an incorrect group sharing charges reference.
INCORRECT_GSD_PARENT_REF	The input XML file includes an incorrect group sharing discounts reference.
INCORRECT_GSP_PARENT_REF	The input XML file includes an incorrect group sharing profiles reference.
PROCESS_IS_RUNNING	The input XML file is either already loaded or currently being loaded by another **pin_cmt** instance.
SQL_ERROR	Internal **pin_cmt** error.

Testing the Imported Data

You test the data to verify that the objects have been created correctly and that record pointers are consistent.

Use any of the following tools to validate the data in your BRM database:

- The **testnap** utility and Object Browser. See "Using testnap and Object Browser to Validate the Database".
- Billing Care. See "Using Billing Care to Validate Data".
- SQL. See "Using SQL to Validate Data".

> **Note**
>
> Before performing the initial test conversion, prepare a test database by using the **pin_setup** script, and then create a backup of the database. Then load the current price list and create another backup. This saves time because it is easier to reload a database than to create one.

Using testnap and Object Browser to Validate the Database

Use **testnap** to verify that the following objects have been created correctly in the BRM database. Use Object Browser to look at the contents of each new object in the BRM database.

- **/account**
- **/bill**
- **/item**
- **/event**
- **/service**
- **/group**
- **/payinfo**
- **/billinfo**
- **/balance_group**
- **/device**
- **/device/num**
- **/device/sim**

> **Note**
>
> Print the results of your **testnap** commands and match the objects in the list. This allows you to make any necessary notes.

For information on how to use **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Reference*.

Validating /account Objects

To validate **/account** objects:

1. Use **testnap** or Object Browser to display the data in the object.
2. Examine each field to ensure the data in the field matches the input data file.
3. Verify that all the charge offers and services owned by this account are present.
4. Verify that all of the balances for this account are correct.

Validating /bill, /item, /event, /service, and /payinfo Objects

To validate these objects:

1. Find the POIDs of these objects in the **/account** object.
2. Use **testnap** or Object Browser to display the data for each of these objects.
3. Examine each field in the object to ensure that the data contained in the field matches the data in the input data file.
4. Ensure that these objects reference the correct **/account** object.

Using Billing Care to Validate Data

To validate data, verify the following:

- You can retrieve account data without any errors.
- You can update an account without any errors.
- You can change payment methods successfully. For example, change an account payment method from credit card to invoice and back to credit card (use the **answer_s** and **answer_b** daemons, if necessary).
- If parent-child billing data was converted, verify that the parent-child grouping works correctly. To check this, do the following:
  - Change some of the existing child accounts to orphan accounts, and some of the existing orphan accounts to child accounts.
  - Add a few arbitrary hierarchies.
  - Move accounts to and from the parent accounts and verify that there are no errors.

Using SQL to Validate Data

You can use SQL to check the various record counts in BRM.

> **Note**
>
> BRM contains the root account, which increases the number of accounts by 1. Remember to consider this at the time of validation.

Verify the following:

- The total number of accounts created is equal to the total number of accounts converted from your legacy system. Use the following SQL statement:

  ```
  select count(*) from ACCOUNT_T
  ```

- The total number of account name and address records is equal to the total number of accounts converted from your legacy system. Use the following SQL statement:

  ```
  select count(*) from ACCOUNT_NAMEINFO_T
  ```

> **Note**
>
> If your implementation has more than one name and address type, you need to take this into account.

- The total number of bill objects is equal to the total number of accounts converted from your legacy system. This number must equal the number of records in the ACCOUNT_T table. Use the following SQL statement:

  ```
  select count(*) from BILL_T
  ```

- The total number of payment objects is equal to the total number of accounts converted from your legacy system. This number should equal the number of records in the ACCOUNT_T table. Use the following SQL statement:

  ```
  select count(*) from PAYINFO_T
  ```

  Also verify that the total records in the PAYINFO_INV_T, PAYINFO_CC_T, and PAYINFO_DD_T tables match the number of records in the ACCOUNT_T table. Use these SQL statements:

  ```
  select count(*) from PAYINFO_INV_T
  select count(*) from PAYINFO_CC_T
  ```

- The total number of **/profile** objects (if created) is equal to the total number of accounts converted from your legacy system: this number should equal the number of records in the ACCOUNT_T table. Use the following SQL statement:

  ```
  select count(*) from PROFILE_custom_table_T
  ```

  where *custom_table* is the implementation-unique identifier you choose; for example, company name.

- The total number of child accounts whose bill units are nonpaying matches the number of rows in the GROUP_BILLING_MEMBERS_T table. Use the following SQL statement:

  ```
  select count(*) from GROUP_BILLING_MEMBERS_T where
  object_type = '/account'
  ```

- The total number of parent accounts matches the number of rows in the GROUP_T table. Use the following SQL statement:

  ```
  select count(*) from GROUP_T where poid_type= '/group/billing'
  ```

- The total number of parent accounts matches the number of rows in the GROUP_PERMITTEDS_T table. Use the following SQL statement:

  ```
  select count(*) from GROUP_PERMITTEDS_T where type = '/account'
  ```

6

Migrating Legacy Data into BRM Table Partitions

This chapter describes how to load data from legacy billing systems into Oracle Communications Billing and Revenue Management (BRM) table partitions.

Topics in this document:

Before you read this chapter, you should be familiar with the following:

- Partitioning. See "Partitioning Tables" in *BRM System Administrator's Guide*
- Conversion Manager. See "Understanding Conversion Manager"

About Migrating Legacy Data into Table Partitions

To load legacy data into BRM table partitions, do the following:

- Create back-dated partitions for storing legacy data. See "About Your Partitioning Scheme".
- Configure Conversion Manager to encode a creation timestamp into specified object POIDs. See "About Making Conversion Manager Aware of Partitions".

About Partitioning

Some BRM tables, such as the BILL_T table, store large amounts of data, which can slow BRM search times. Dividing large tables into smaller, more manageable chunks, called partitions, makes it easier for BRM to find data because it needs to search only a single partition for data rather than an entire table.

In BRM, you create table partitions based on dates and divided by a specified interval: month, week, or day. For example, you could set one partition for May 1 to May 31, the subsequent one for June 1 to June 30, and so on. Objects are stored in partitions according to their creation date and time. For example, **/event** objects created in May 2009 could be stored in the May 2009 partition of the EVENT_T table.

About Your Partitioning Scheme

Before you begin loading legacy data into BRM, you must determine your partitioning scheme, including which tables to partition and how many months of legacy data to load into BRM. For example, you might decide to load only the last five months of your billing data into the BRM system. In this case, you would create five back-dated partitions with a monthly interval for the BILL_T table.

After determining your partitioning scheme, you must create the following:

- **Back-dated table partitions for your legacy data**. For example, if you are loading legacy billing and item data into BRM, create back-dated table partitions for the **/bill** storable class (BILL_T table) and the **/item** storable class (ITEM_T table). Your back-dated tables must also comply with the following restrictions.
  - The oldest back-dated partition that you create must have a start date prior to the creation date of the oldest legacy object you are loading into BRM. For example, if the oldest object that you are loading has a creation date of March 8, 2005, you must create a back-dated table partition that starts on or before March 7, 2005.
  - The most recent back-dated partition that you create must have an end date that is at least one day prior to the BRM installation date. For example, if you install BRM on August 15, 2009, the most recent back-dated partition must end on or before August 14, 2009.
- **Future-dated partitions for data created on your new BRM system**. Create future-dated partitions for any storable classes you would like partitioned. The first future-dated partition must start two days after the BRM installation date. For example, if you install BRM on August 15, 2009, the first partition's start date must be August 17, 2009, or later.

For example, assume you install BRM on August 15, 2009, and the oldest billing object in your legacy system has a creation date of January 20, 2009. In this example, you would create seven back-dated partitions and one or more future-dated partitions for your **/bill** storable class (BILL_T table) as shown in Table 6-1.

Table 6-1 Partitioning Scheme Example

Partition	Date Range for Each BILL_T Partition
1	January 15, 2009, through February 14, 2009
2	February 15, 2009, through March 14, 2009
3	March 15, 2009, through April 14, 2009
4	April 15, 2009, through May 14, 2009
5	May 15, 2009, through June 14, 2009
6	June 15, 2009, through July 14, 2009
7	July 15, 2009, through August 14, 2009
8	August 17, 2009, through September 16, 2009
9	September 17, 2009, through October 16, 2009
10	**partition_last**

In this example, objects are loaded into partitions as shown below:

- Objects with creation timestamps prior to February 15, 2009, are automatically loaded into Partition 1.
- Objects with creation timestamps on or after February 15, 2009, and prior to March 15, 2009, are loaded into Partition 2.
- Objects with creation timestamps after August 14, 2009, and prior to August 17, 2009, are loaded into Partition 8.

> **Note**
>
> If an object's creation timestamp is not covered by one of the partition ranges, the object is automatically loaded into the next higher-range partition.

- Objects with creation timestamps after October 16, 2009, are loaded into partition 10 (**partition_last**).

> **Note**
>
> If an object's creation timestamp is not covered by one of the partition ranges and a higher-range partition does not exist, the object is automatically loaded into **partition_last**.

About Making Conversion Manager Aware of Partitions

Conversion Manager can now encode timestamps into the object POIDs of partitioned and purgeable storable classes, allowing the object to be loaded into the correct table partition. For example, if a legacy bill object was created in May, Conversion Manager encodes the creation timestamp into the **/bill** object's POID. The **/bill** object can then be loaded into the BILL_T table's May 2009 partition.

About the Timestamps Encoded in Object POIDs

The timestamp encoded in object POIDs can be either:

- The creation timestamp passed in by the legacy system. In this case, you must pass the object creation timestamp in the **CrtT** element of the input XML file.
- The system time when you run Conversion Manager. It does not use the timestamp from **pin_virtual_time**.

You must make sure that the creation time is applied consistently to all related objects. For example:

- When you migrate unpaid bills and open items from your legacy billing system to BRM, the bill objects and their associated item objects must all use the legacy object's creation time. Otherwise, the billing and invoicing utilities will not find all items associated with a bill.
- When you migrate balances from your legacy system to BRM, the account objects and their associated balances must all use the legacy object's creation time.

Conversion Manager Tasks for Partitioned Storable Classes

When Conversion Manager is configured to encode timestamps into object POIDs, it performs these additional tasks:

- Determines which storable classes are partitioned and purgeable by reading the BRM data dictionary. In the data dictionary, partitioned and purgeable storable classes have their IS_PARTITIONED field set to **1** and their PARTITION_MODE field set to **Finite**.
- Reserves a set of POIDs for each partitioned, purgeable storable class and a single set of POIDs for all nonpartitioned storable classes. For example, if the **/bill** and **/item** storable classes are partitioned and purgeable, Conversion Manager reserves three sets of POIDs:

one for **/bill** objects, one for **/item** objects, and one for objects of all nonpartitioned storable classes.

- Assigns POIDs to objects based on the storable class type:
  - If the storable class is partitioned and purgeable, Conversion Manager assigns a POID from the storable class's reserve of partitioned POIDs and encodes the timestamp. It uses either the timestamp passed in from the legacy system or the system time when you ran Conversion Manager.
  - If the storable class is not partitioned, Conversion Manager assigns a POID from the reserve of nonpartitioned POIDs.

> **Note**
>
> If the set of reserved POIDs is already depleted, Conversion Manager reserves a new set of POIDs before assigning one to an object.

The object is then loaded into a partition according to the storable class type and the timestamp encoded in the object POID.

During the loading phase, if no corresponding partition is available for the timestamp encoded in the object POID, the object is loaded into the next higher-range partition. If no higher-range partition is available, the object is loaded into **partition_last**.

If a storable class *is not* partitioned, the object is loaded directly into the target table.

About Configuring Conversion Manager to Encode Timestamps in POIDs

To configure Conversion Manager to encode timestamps in object POIDs, you specify the following in the Conversion Manager **Infranet.properties** file:

- The number of POIDs to reserve at a time.
- Which timestamp to encode into POIDs.

For more information, see "Configuring Conversion Manager for Partitioning".

Setting Up Your System to Load Legacy Data into Table Partitions

To set up your system to load data from legacy billing systems into BRM table partitions:

1. Make sure partitioning is enabled in your BRM system by doing either or both of the following:
   - When installing BRM, choose to partition your tables.
   - After installing BRM, convert nonpartitioned storable classes to partitioned storable classes.

   For more information, see "Partitioning Tables" in *BRM System Administrator's Guide*.
2. Determine the oldest object for each storable class that will be partitioned in the BRM database. The oldest partition's start date must be prior to the oldest object's creation date for that storable class.
3. Create partitions with the **partition_utils** utility. See "Creating Partitions for Your Legacy Data".

4. Configure Conversion Manager for partitioning. See "Configuring Conversion Manager for Partitioning".
5. Include the legacy object's creation timestamp in the Conversion Manager input XML file. See "Passing Object Creation Timestamps in the Input XML File".

Creating Partitions for Your Legacy Data

To create back-dated partitions, enter the following command:

```
partition_utils  -o add  -t realtime  -s start_date  -u month|week|day  -q quantity
                [-c storable_class]  [-w width]  -b
```

where:

- *start_date* specifies the starting date for the first partition. The format is *MMDDYYYY*. The start date must be before the BRM installation date. The oldest partition's start date must be before the oldest object's creation date.
- *quantity* specifies the number of partitions to add.
- *storable_class* specifies the storable class to store in the partition. The default is **/event**.
- *width* specifies the number of units in a partition, such as 3.

You must also create future-dated partitions for objects generated by your new BRM system. For more information on how to create future-dated partitions, see "Partitioning Database Tables" in *BRM System Administrator's Guide*.

Configuring Conversion Manager for Partitioning

By default, Conversion Manager does not encode timestamps in object POIDs. You configure Conversion Manager to do so through its **Infranet.properties** file.

To configure Conversion Manager for partitioning:

1. Open the *BRM_home***/apps/CMT/Infranet.properties** file in a text editor. *BRM_home* is the directory in which the BRM server software is installed.
2. Set the entries for creating partitioning-aware object POIDs:
   - Use **infranet.cmt.timestampvalidation** to specify which timestamp to encode in generated POIDs. See "Configuring Which Timestamp to Encode".
   - Use **infranet.cmt.noofrecords** to specify the number of POIDs to reserve for partitioned POIDs. See "Configuring the Number of POIDs to Reserve".
3. Save and close the file.

Configuring Which Timestamp to Encode

The timestamp encoded in object POIDs can be either the object creation time passed in from the legacy system or the system time when you run **pin_cmt**.

> **Note**
>
> If you specify to use the system time, Conversion Manager loads all of the legacy data into one partition. You must configure and size the appropriate partition to store the volume of data that will be loaded into it.

You specify which timestamp to encode by using the following **Infranet.properties** file entry:

```
infranet.cmt.timestampvalidation = Value
```

where *Value* is one of the following:

- **0**: The timestamp is set to the creation time passed in the input XML file. If a creation time is not passed in, Conversion Manager assigns the system time when you run **pin_cmt**.
- **1**: The timestamp is set to the creation time passed in the XML file only. If a creation time *is not* passed in, Conversion Manager generates an error. This setting ensures that all partitioned classes include the original creation timestamp. This is the default.
- **2**: The timestamp is set to the system time when you run **pin_cmt**. If a creation timestamp *is* passed in, Conversion Manager generates an error. This ensures that all partitioned classes are loaded with the system time.

> **Note**
>
> This parameter cannot enforce validations across multiple runs of Conversion Manager. Loading some objects with the system time and other objects with the object creation time could cause system inconsistencies and business operations to fail.

Configuring the Number of POIDs to Reserve

Conversion Manager reserves a set of POIDs for each storable class type that is partitioned and reserves another set of POIDs for all other nonpartitioned storable classes.

To configure the number of POIDs to reserve:

- For partitioned storable classes, use the following entry:

  ```
  infranet.cmt.noofrecords = Number
  ```

  where *Number* specifies the number of POIDs to reserve. The default is **1**.

  Conversion Manager reserves the specified number of POIDs for each storable class that is partitioned.

- For nonpartitioned storable classes, use the following entries:

  ```
  infranet.cmt.noofrecords = Number
  infranet.cmt.avgnoofservices = Services
  infranet.cmt.avgnoofdevices = Devices
  ```

  where:

  - *Number* specifies the number of POIDs to reserve. The default is **1**.
  - *Services* specifies the average number of services in your accounts. The default is **5**.
  - *Devices* specifies the average number of devices in your accounts. The default is **2**.

  Conversion Manager multiplies these three values (*Number* ***** *Services* ***** *Devices*) to compute the total number of POIDs to reserve for nonpartitioned storable classes.

For example, assume you partition the **/bill** and **/item** storable classes and the **Infranet.properties** file includes the following entries:

```
infranet.cmt.noofrecords = 1000
infranet.cmt.avgnoofservices = 5
infranet.cmt.avgnoofdevices = 3
```

In this example, Conversion Manager reserves the following POIDs:

- 1,000 POIDs for **/bill** objects.
- 1,000 POIDs for **/item** objects.
- 15,000 POIDs for all other objects.

Passing Object Creation Timestamps in the Input XML File

The Conversion Manager input XML file includes a **CrtT** element for all storable classes. If you want to encode the legacy object's creation time in your POIDs, you must pass the object's creation time in the **CrtT** element of the input XML file.

When passing the legacy object's creation time, make sure:

- The object creation timestamp is correct. Conversion Manager does not perform any validations on values passed in the **CrtT** element.
- Objects using the legacy creation time are not combined in the same partitioned, purgeable table as objects using the system time.

7

Conversion Manager Utilities

This chapter provides reference information for Oracle Communications Billing and Revenue Management (BRM) Conversion Manager utilities.

Topics in this document:

cmt_mta_cycle_fees

> **Note**
>
> The **cmt_mta_cycle_fees** utility is run internally by the **pin_cmt** utility. Do not run this utility by itself.

The **cmt_mta_cycle_fees** utility applies cycle forward fees to BRM accounts that have been deployed by the **pin_cmt** utility. See "Understanding Conversion Manager" and "Loading Legacy Data into the BRM Database".

> **Note**
>
> - Conversion Manager is an optional component, not part of base BRM.
> - To connect to the BRM database, the **cmt_mta_cycle_fees** utility needs a configuration file in the directory from which you run the utility.

Location

*BRM_home***/apps/cmt**

where *BRM_home* is the directory in which the BRM server software is installed.

Syntax

```
cmt_mta_cycle_fees -stage_id stage_ID -cycle_dom day_of_month
```

Parameters

-stage_id *stage_ID*
The stage ID used when the data was imported.

-cycle_dom *day_of_month*
The billing day of month for the accounts that need cycle fees applied.

Results

The **cmt_mta_cycle_fees** utility notifies you when it runs successfully. Otherwise, look in the **pin_mta.pinlog** file in *BRM_home***/apps/cmt** for errors.

pin_cmt

Use the **pin_cmt** utility to load legacy data in XML format into your BRM database. See "Understanding Conversion Manager" and "Loading Legacy Data into the BRM Database".

> **Note**
>
> - Conversion Manager is an optional component, not part of base BRM.
> - To specify a database connection, edit the **pin_cmt Infranet.properties** file in *BRM_home***/apps/cmt**.

Location

*BRM_home***/apps/cmt**

Syntax

```
pin_cmt -import -file XML_input_data_file stage_ID|
        -import_custom -file XML_input_data_file stage_ID|
        -recovery load batch_ID|
        -deploy DOM stage_ID
```

Parameters

-import -file *XML_input_data_file stage_ID*
Imports the specified data file into the BRM database.

> **Note**
>
> Ensure that there are no extra spaces in the input XML file. If you need to indent text in the XML file, use the TAB key to add space.

-import_custom -file *XML_input_data_file stage_ID*
Imports data that uses new storable classes into the BRM database.

-recovery load *batch_id*
Reloads data after a failed load process. The batch ID is recorded in the **cmt.pinlog** file in *BRM_home***/apps/cmt.**

-deploy *DOM stage_ID*
Deploys data. DOM is the billing cycle's day of the month. *stage_ID* is the stage ID used when the data was imported.
In this example, accounts with a billing day of month of 15 and stage ID of 100 are deployed:

```
pin_cmt -deploy 15 100
```

Results

The **pin_cmt** utility notifies you when it runs successfully. Otherwise, look in the **cmt.pinlog** file in *BRM_home*/**apps/cmt** for errors. In addition, look for errors in the log file specified in the **pin.conf** file for the **cmt_mta_cycle_fees** utility.