Oracle® Financial Services Analytical Applications Data Model Utilities

User Guide

Release 7.3

Part No. E35138-01

June 2014

ORACLE®

Oracle Financial Services Analytical Applications Data Model Utilities User Guide, Release 7.3

Part No. E35138-01

Primary Author: Aneesh Kurian

Contributor: Aravind Venketaraman, Lijo Kattakayam

Contents

Send Us Your Comments

Oracle Financial Services Analytical Applications Data Model Utilities User Guide, Release 7.3

Part No. E35138-01

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document. Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Send your comments to us using the electronic mail address: financialservices_ww@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

Intended Audience

Welcome to Release 7.3 of the *Oracle Financial Services Analytical Applications Data Model Utilities User Guide.*

See Related Information Sources on page viii for more Oracle product information.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Structure

1 Introduction

This document contains various chapters related to data model utilities and data loaders available within Oracle Financial Services Analytical Applications (OFSAA). The four chapters present in this document are: Object Management, Utilities, Data Loaders, and Mapping Export in Metadata Browser.

2 Object Management

This chapter details the steps involved in adding various client data objects into the model.

3 Utilities

This chapter details the steps involved in executing various data model utilities that are available within OFSAA.

4 Data Loaders

This chapter details the steps involved in executing various data loaders that are available within OFSAA. Data loaders move data from staging layer to processing layer.

5 Mapping Export in Metadata Browser

6 SCD Configuration

Related Information Sources

- Oracle Financial Services Analytical Applications Infrastructure (OFSAAI) Installation and Configuration Guide
- Oracle Financial Services Analytical Applications Infrastructure User Guide
- Oracle Financial Services Cash Flow Engine Reference Guide
- Oracle Financial Services Asset Liability Management (OFSALM) User Guide
- Oracle Financial Services Profitability Management (OFSPM) User Guide
- Oracle Financial Services Funds Transfer Pricing User Guide

1

Introduction

This document contains various chapters related to data model utilities and data loaders available within Oracle Financial Services Analytical Applications (OFSAA). The four chapters present in this document are: Object Management, Utilities, Data Loaders, and Mapping Export in Metadata Browser.

This chapter covers the following topics:

- List of Acronyms Used in the Document

List of Acronyms Used in the Document

Acronym	Description
AAI	Analytical Applications Infrastructure
ALM	Asset Liability Management
AMHM	Attributes, Members and Hierarchy Management
COA	Chart Of Accounts
F2T	File to Table
FDM	Financial Data Manager
GL	General Ledger
GTT	Global Temporary Table

Acronym	Description
ICC	Information Command Center
INFODOM	Information Domain
IP	Internet Protocol
OFS	Oracle Financial Services
OFSA	Oracle Financial Services Applications
OFSAA	Oracle Financial Services Analytical Applications
OFSAAI	Oracle Financial Services Analytical Applications Infrastructure
PFT	Profitability
PL/SQL	Procedural Language /Structured Query Language
T2T	Table to Table
TP	Transfer Pricing
UDP	User-Defined Property
UI	User Interface

2

Object Management

This chapter details the steps involved in adding various client data objects into the model.

This chapter covers the following topics:

- Adding Dimension Tables and Key Dimension (Leaf) Registration
- Adding Custom Instrument Tables
- Adding Custom Transaction Tables
- Adding Custom Lookup Tables
- Object Registration And Validation
- Defining Alternate Rate Output Columns
- User Defined Properties
- Modifying the Precision of Balance Columns In Ledger Stat

Adding Dimension Tables and Key Dimension (Leaf) Registration

The following section details the process in which users can add custom key dimensions to the OFSAA application. Users can view the registered dimension within the AMHM screens. Also, users can add members and hierarchies for the dimension through AMHM screens.

Registering a new Key Dimension (called as Leaf in OFSA 4.5) requires the following steps:

- Add a set of dimension tables to store leaf values in ERwin model.
- Add the key dimension column to required Entities in ERwin model.
- Assign the Processing Key Column Property (Key Dimension Columns only).
- Upload the model.

- Register the Key Dimension.
- Modify Unique indexes (Key Leaf Dimension only).
- Validate tables.

Each of these steps is discussed in detail in the following sections.

Adding Dimension Tables

Adding Key Dimension Tables

Each key dimension contains a set of the following tables:

- DIM_<DIMENSION>_B - Stores leaf and node member codes within the dimension.
- DIM_<DIMENSION>_TL - Stores names of leaf and node and their translations.
- DIM_<DIMENSION>_ATTR - Stores attribute values for the attributes of the dimension.
- DIM_<DIMENSION>_HIER - Stores parent-child relationship of members and nodes that are part of hierarchies.

> **Note:** Replace <DIMENSION> with the keyword representing the key dimension.

Seeded key dimension tables are present in 'Fusion – Dimensions' subject area within the ERwin model. The above tables need to be created for the new dimension. For more information on creating dimension tables in ERwin, see *leaflet (AddingAndCustomizingLeaf.pdf)*.

> **Note:** For ease of use, user can copy an existing set of dimension tables eg, for ORG_UNIT dimension and rename the tables (in both physical and logical view) to represent the new dimension.

Table structure of one of the seeded key dimension is given below with remarks on how this can be used as the basis for modeling new key dimensions.

DIM_ORG_UNIT_B

Stores the ID of the members (leaf and nodes) of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
ORG_UNIT_ID	Organization Unit ID	NUMBER(14)	NOT NULL	Leaf column which stores the id for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
ORG_UNIT_DISPLAY_CODE	Organization Unit Display Code	NUMBER(14)	NULL	Leaf column which stores the display code for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
ENABLED_FLAG	Enabled Flag	VARCHAR2(1)	NOT NULL	Store if the item is enabled or not	Internally used and hence should be retained in the same form within the new dimension table.
LEAF_ONLY_FLAG	Leaf or Node Flag	VARCHAR2(1)	NOT NULL	Indicates if the member is leaf only or not	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
DEFINITION_LANGUAGE	Definition Language	VARCHAR2(4)	NOT NULL	Language that is used to define	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
ORG_UNIT_CODE	ORG_UNIT_CODE	VARCHAR2(20)	NULL	This column is used by staging and contains the alpha-numeric codes for each dimension member. Staging dimension table contains unique alpha-numeric codes and a unique numeric identifier is generated while loading into Fusion dimension table.	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.

DIM_ORG_UNIT_TL

Stores the names and descriptions of the members (leaf and nodes) of the dimension in various languages.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
LANGUAGE	Language	VARCHAR2(4)	NOT NULL	Language	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
ORG_UNIT_ID	Organization Unit ID	NUMBER(14)	NOT NULL	Leaf column which stores the id for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
ORG_UNIT_NAME	Organization Unit Name	VARCHAR2(150)	NOT NULL	Leaf column which stores the name for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
DESCRIPTION	Description	VARCHAR2(255)	NULL	Description of an Item	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

DIM_ORG_UNIT_ATTR

Stores the values of the attributes of the members (leaf and nodes) of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
ORG_UNIT_ID	Organization Unit ID	NUMBER(14)	NOT NULL	Leaf column which stores the id for the organisation unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
ATTRIBUTE_ID	Attribute ID	NUMBER(22)	NOT NULL	Stores attribute id number for a member of a dimension	Internally used and hence should be retained in the same form within the new dimension table.
DIM_ATTRIBUTE_NUMERIC_MEMBER	Numeric Dimension Value	NUMBER(22)	NULL	This field stores the number values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
DIM_ATTRIBUTE_VARCHAR_MEMBER	Varchar Dimension Value	VARCHAR2(30)	NULL	This field stores the varchar values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
NUMBER_ASSIGN_VALUE	Numeric Value Of A Member	NUMBER(22)	NULL	This field stores the number values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
VARCHAR_ASSIGN_VALUE	Varchar Member Value	VARCHAR2(1000)	NULL	This field stores the varchar values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
DATE_ASSIGN_VALUE	Date Value	DATE	NULL	Date value that is assigned	Internally used and hence should be retained in the same form within the new dimension table.

DIM_ORG_UNIT_HIER

Stores the parent-child relationship of various nodes and leaf within hierarchies of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
HIERARCHY_ID	Hierarchy ID	NUMBER(10)	NOT NULL	Unique Id that is generated for every hierarchy that is created	Internally used and hence should be retained in the same form within the new dimension table.
PARENT_ID	Parent ID	NUMBER(14)	NOT NULL	Column that store the id of the child member	Internally used and hence should be retained in the same form within the new dimension table.
CHILD_ID	Child Member ID	NUMBER(14)	NOT NULL	Store child id number for a dimension	Internally used and hence should be retained in the same form within the new dimension table.
PARENT_DEPTH_NUM	Parent Depth Number	NUMBER(14)	NOT NULL	Stores parent depth number	Internally used and hence should be retained in the same form within the new dimension table.
CHILD_DEPTH_NUM	Child Depth Number	NUMBER(14)	NOT NULL	Stores child depth number	Internally used and hence should be retained in the same form within the new dimension table.
DISPLAY_ORDER_NUM	Display Order Number	NUMBER(14)	NOT NULL	Stores the display order number for the member	Internally used and hence should be retained in the same form within the new dimension table.
SINGLE_DEPTH_FLAG	Single Depth Flag	VARCHAR2(1)	NOT NULL	Indicates if the hierarchy is of single depth or not	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

Adding Simple Dimension Tables

Simple dimensions are created to store CODE and Descriptions. These tables are used by the User Interfaces to list values in drop downs / radio buttons, and so on.

Each simple dimension contains a set of the following tables:

- CD table – Stores the members for a simple dimension.
- MLS table – Stores the members' multi lingual description.

If you use simple dimensions where _CD column is VARCHAR2, you will need to classify the tables as follows:.

Example

FSI_ACCUMULATION_TYPE_CD, FSI_BILLING_METHOD_CD

The CD table should be classified as

295	Codes User Defined (base tbl)
198	Codes Reserved (base tbl)

The MLS table should be classified as

296	MLS Descriptions User Defined
197	MLS Descriptions Reserved

Table structure of one of these seeded simple dimensions is given in the following section with remarks on how this can be used as the basis for modeling new simple dimensions.

FSI_<DIM>_CD

Stores the ID of the members (leaf and nodes) of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
DIM_CD	Dimension Code	NUMBER(5)	NOT NULL	Leaf column which stores the code for the dimension.	Stores the Dimension Code.
LEAF_ONLY_FLAG	Leaf or Node Flag	VARCHAR2(1)	NOT NULL	Indicates if the member is leaf only or not	Internally used and hence should be retained in the same form within the new dimension table.
ENABLED_FLAG	Enabled Flag	VARCHAR2(1)	NOT NULL	Store if the item is enabled or not	Internally used and hence should be retained in the same form within the new dimension table.
DEFINITION_LANGUAGE	Definition Language	VARCHAR2(4)	NOT NULL	Language that is used to define	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
<DIM>_DISPLAY_CD	Dimension Display Code	VARCHAR2()	NULL	Leaf column which stores the display code for the dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained. The length of this column is customisable.

FSI_<DIM>_MLS

Stores the members' multi lingual description.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
DIM_CD	Dimension Code	NUMBER(5)	NOT NULL	Leaf column which stores the code for the dimension.	Stores the Dimension Code.
LANGUAGE	Language	VARCHAR2(3)	NOT NULL	Language	Internally used and hence should be retained in the same form within the new dimension table.
<DIM>	Dimension	VARCHAR2(40)	NOT NULL	Name of the Dimension	Stores the name of the Dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
DESCRIPTION	Description	VARCHAR2(255)	NULL	Description of an Item	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

Example for FSI_<DIM>_CD table::

```
CREATE TABLE <XXXXX>_FSI_<DIM>_CD   -- ACME_FSI_ACCT_STATUS_CD
(<DIM>_CD            NUMBER(5)    -- ACCT_STATUS_CD
,LEAF_ONLY_FLAG      VARCHAR2(1)
,ENABLED_FLAG        VARCHAR2(1)
,DEFINITION_LANGUAGE VARCHAR2(4)
,CREATED_BY          VARCHAR2(30)
,CREATION_DATE       DATE
,LAST_MODIFIED_BY    VARCHAR2(30)
,LAST_MODIFIED_DATE  DATE
<dim>_display_CD     VARCHAR2()
);
```

Example for FSI_<DIM>_MLS table::

```
CREATE TABLE <XXXXX>_FSI_<DIM>_MLS   -- ACME_FSI_ACCT_STATUS_CD
(<DIM>_CD            NUMBER(5)    -- ACCT_STATUS_CD
,LANGUAGE            VARCHAR2(3)
,<DIM>               VARCHAR2(40)  -- ACCT_STATUS
,DESCRIPTION         VARCHAR2(255)
,CREATED_BY          VARCHAR2(30)
,CREATION_DATE       DATE
,LAST_MODIFIED_BY    VARCHAR2(30)
,LAST_MODIFIED_DATE  DATE
);
```

Adding Dimension Column To Required Objects

Dimension column can be added to the following set of Client Data Objects:

- Tables classified as 'Instruments' and 'Instrument Profitability'
- Tables classified as 'Transaction Profitability'
- *Ledger Stat* table.

Dimension can be of the types – Ledger Only or Both. If the dimension is classified as 'Ledger Only', the dimension column needs to be added only to *Ledger Stat* table. If the dimension is classified as 'Both', the dimension column needs to be added to *Ledger Stat* table and other tables classified as Instruments and Transactions.

For adding key dimension column to tables that are classified as 'Instruments' and 'Instrument Profitability', add the column to LEAF_COLUMNS super-class table.

For adding key dimension column to tables that are classified as 'Transaction Profitability', add the column to TRANS_LEAF_COLUMNS super-class table.

For adding key dimension column to *Ledger Stat* table, add the column to LEDGER_LEAF_COLUMNS super-class table.

> **Note:** Columns of super-class tables that are linked to sub-class table are rolled down to the sub-class table during 'Model Upload' operation.

Assigning Processing Key Property

'Processing Key' is a column level User Defined Property (UDP) in ERwin model. This property can have two values – Yes or No. Only those objects where the column was added to the unique index are affected.

For tables classified as 'Transaction Profitability, this property needs to be set as 'Yes' for one or more of the key dimension columns.

For *Ledger Stat* table, this property needs to be set as 'Yes' for all key dimension columns.

Assigning Processing Key Property is not required for Simple Dimension.

Uploading ERwin Model

ERwin model with the above changes needs to be uploaded in OFSAAI environment. Uploading the model creates these additional tables and sets these properties within the atomic schema.

After upload, user can verify the changes in the schema as well as query OFSAAI metadata tables like REV_COLUMN_PROPERTIES for viewing properties assigned to each column.

For more information on data model upload process, see *OFSAAI User Guide*.

Leaf Registration

Oracle Financial Services Analytical Applications Infrastructure (OFSAAI) provides an Leaf Registration procedure to add the new Key Dimension Column to the Dimensions metadata registry (REV_DIMENSIONS_B, REV_DIMENSIONS_TL).

Leaf Registration Procedure

This procedure performs the following:

- Registers key and simple dimension.
- Invalidates all Client Data Objects when key dimension is registered.

Executing Leaf Registration Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires 19 parameters. The syntax for calling the procedure is:

```
function  rev_leaf_registration(batch_run_id  varchar2,
                            mis_date varchar2
                              memDataType varchar2,
                              dimName varchar2,
                              description varchar2,
                              memberBTableName varchar2,
                              memberTLTableName varchar2,
                              hierarchyTableName varchar2,
                              attributeTableName varchar2,
                              memberCol varchar2,
                              memberDispCodeCol varchar2,
                              memberNameCol varchar2,
                              memberDescCol varchar2,
                              dimTypeCode varchar2,
                              simpleDimFlag varchar2,
                              keyDimFlag char,
                                   writeFlag  varchar2,
                              catalogTableType char,
                                   flattenedTableName varchar2)
```

- batch_run_id : any string to identify the executed batch.
- mis_date : in the format YYYYMMDD.
- memDataType : member data type of Dimension as in NUMBER,VARCHAR2,CHAR.
- dimName : name of the dimension to be added (less than 21 chars).
- description : description of the dimension (less than 255 chars).
- memberBTableName : Member Base Table Name input as either null or a value with suffix '_CD' or '_B'.
- memberTLTableName : Member TL Table Name input as either null or name of the table.
- hierarchyTableName : Hierarchy Table Name input as either null or name of the table.
- attributeTableName : Attribute Table Name input as either null or name of the table.
- memberCol : Member Column Name input as either null or name of the column.
- memberDispCodeCol : Member Display Code Column Name input as either null or name of the column.
- memberNameCol : Member Name Column input as either null or name of the column.
- memberDescCol : Member Description Column input as either null or name of the

column.

- dimTypeCode : Code for the dimension Type as in 'PROD for product type', 'ORGN for Organizational Unit', 'CCOA for Common Chart of Accounts', 'FINELE for Financial Element', 'GL for General Ledger Account', 'OTHER for any other type'.

  All user defined dimensions will have DIMENSION_TYPE_CODE as 'OTHER'. User defined dimensions which are product related will have DIMENSION_TYPE_CODE as 'PROD'.

- simpleDimFlag : 'Y' or 'N' to determine Simple Dimension.

  Simple dimensions are created to store CODE and Descriptions. These tables are used by the User Interfaces to list values in drop downs / radio buttons, and so on. Simple dimensions are not reverse populated.

  Example
  Country, Currencies, Customer Type.

- keyDimFlag : 'Y' or 'N' to determine Key Dimension.

  Key dimensions are dimensions which get reverse populated to the legacy tables.

  Example
  Product, Org Unit, General Ledger.

- writeFlag : 'Y' or 'N' to determine whether Dimension should appear in drop down list in Dimension Management > Members.

- catalogTableType : 'L' or 'B' to determine table type for key dimensions.

- flattenedTableName : Flattened Table Name input as either null or name of the table.

Example for Key Dimension:

```
Declare
     num number;
Begin
    num := rev_leaf_registration('BATCH_NO_01',
                                 '20101216',
                                 'NUMBER',
                                 'PRODUCT_1',
                                 'Cost Transfer Product Type ID',
                                 'DIM_PRODUCT_1_B',
                                 'DIM_PRODUCT_1_TL',
                                 'DIM_PRODUCT_1_HIER',
                                 'DIM_PRODUCT_1_ATTR',
                                 'PRODUCT_1_ID',
                                 'PRODUCT_1_DISPLAY_CODE',
                                 'PRODUCT_1_NAME',
                                  'DESCRIPTION',
                                  'PROD',
                                   'N',
                                   'Y',
                                   'Y',
                                   'B',
                                  'FLATTEN_PROD_TABLE');

End;
```

Example for Simple Dimension:

```
Declare
     num number;
Begin
    num := rev_leaf_registration('BATCH_NO_01',
                                 '20101216',
                                 'NUMBER',
                                 'SIMPLE DIMENSION',
                                 'SIMPLE DIMENSION DESC',
                                 'FSI_DIM_SIMPLE_CD',
                                 'FSI_DIM_SIMPLE_MLS',
                                 'null',
                                 'null',
                                 'SIMPLE_CD',
                                 'SIMPLE_DISPLAY_CODE',
                                 'SIMPLE_NAME_Dim',
                                  'SIMPLE_DESCRIPTION',
                                  'OTHER',
                                   'Y',
                                   'N',
                                   'Y',
                                   'B',
                                  'FLATTEN_PROD_TABLE');

End;
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list

- IP address:- Select the IP address from the list
- Rule Name:- *batch_leaf_registration*
- Parameter List:- Member Data type , Dimension Name, Dimension Description, Member Base Table Name, Member Translation Table Name, Hierarchy Table Name, Attribute Table Name, Member Column , Member Display Code Column, Member Name Column, Member Description Column , Dimension Type Code , Simple Dimension Flag , Key Dimension Flag , writeFlag, Catalog Table Type , Flatten Table Name

Modify Unique Indexes

For tables of 'Transaction Profitability' classification, key dimension column can be part of the unique index. If this column is intended to be part of the unique index, alter the unique index in the schema.

For Ledger Stat table, all key dimension columns should form part of the unique index. Hence, alter the unique index in the schema to include this column.

Executing Object Registration Validation

Since leaf registration invalidates all Client Data Objects, Object Registration Validation procedure needs to be executed to validate the required tables. For more information on Executing Object Registration Validation, see Object Registration Validation, page 2-34.

Adding Custom Instrument Tables

Instrument and Account objects are tables storing financial services information about customers and accounts. These are most commonly used objects for OFSAA processing and reporting operations. There are seeded instrument tables that are packaged as part of each OFSAA. You can customize or remove any of them during implementation. In some cases, you might also require to add a custom instrument table.

The following topics are covered in this section:

- Super-class entities
- Steps in creating a custom instrument table
- Setting Table Classifications
- Unique Index
- Object Registration Validation

Super-class Entities

Most instrument tables are used for OFSAA processing. OFSAA processing mandates the instrument table to have a certain set of columns. These columns have been put together in super-class entities. The following are the seeded super-class entities:

- LEAF_COLUMNS – contains the key dimension columns that are part of the Instrument tables.
- BASIC_INSTRUMENT_REQ – contains the basic instrument columns like ID_NUMBER, IDENTITY_CODE etc.
- MULTI_CUR_REQ – contains the columns required for multi-currency processing.
- CASH_FLOW_EDIT_REQ – contains the columns required for Cash flow Edit processing.
- CASH_FLOW_PROC_REQ – contains the columns required for Cash flow processing.
- TP_BASIC_REQ – contains the columns required for Transfer Pricing processing.
- TP_OPTION_COSTING_REQ – contains the columns required for Transfer Pricing Option Cost processing.
- PORTFOLIO_REQ – contains the columns required for Portfolio table classification.
- TRANS_LEAF_COLUMNS – contains the key dimension columns that are part of the transaction tables.
- LEDGER_LEAF_COLUMNS – contains the key dimension columns that are part of the Ledger Stat table.

Instrument table can link to any of the above super-class entities based on its purpose. For example, if the instrument table is used for Cash Flow Processing, then this table should be linked to the following super-class entities:

- BASIC_INSTRUMENT_REQ
- MULTI_CUR_REQ
- LEAF_COLUMNS
- CASH_FLOW_EDIT_REQ
- CASH_FLOW_PROC_REQ

Refer to the following mapping table that specifies the list of super-class entities

required for each table classification:

Type of Client Data Object	Table Classification	List of Super-class entities
Instrument	Instrument	BASIC_INSTRUMENT_REQ LEAF_COLUMNS
Instrument	ALM Standard	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ CASH_FLOW_PROC_REQ
Instrument	TP Cash Flow	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ CASH_FLOW_PROC_REQ TP_BASIC_REQ
Instrument	TP Non-Cash Flow	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ TP_BASIC_REQ
Instrument	TP Option Costing	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ TP_BASIC_REQ TP_OPTION_COSTING_REQ
Instrument	Instrument Profitability	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ
Instrument	Portfolio	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ PORTFOLIO
Transaction	Transaction Profitability	TRANS_LEAF_COLUMNS
Ledger Stat	Ledger Stat	LEDGER_LEAF_COLUMNS

Steps in Creating Custom Instrument Table

The following are the steps involved in creating a custom instrument table:

- Create a new subject area within the ERwin model.
- Move the required super-class tables as part of the subject area.
- Create the custom instrument table in ERwin. Specify logical name, physical name and description for the table. Define any columns that do not come from any of the standard super-class tables as part of the custom instrument table. Specify logical, physical names, domain and other column properties for each column.
- Create subtype relationship between the custom instrument table and various super-class entities.

Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom instrument table:

- Choose Physical View within the ERwin model.
- Go to UDP tab within Table Properties window.
- Specify 'Yes' against required Table Classifications properties.

Once the model is prepared using the above steps, user should upload the ERwin model. After uploading the model, user can check if the custom instrument table has been created in the schema with columns from super-class entities that have been linked to the custom instrument table as well as the columns present in the custom instrument table. Model upload also creates metadata entries within the following Object Registration tables:

- REV_TABLES_B – Contains the list of table names.
- REV_TABLES_TL – contains the list of table display names and descriptions in various languages.
- REV_TAB_COLUMNS – contains the list of column names.
- REV_TAB_COLUMNS_MLS – contains the list of column display names and descriptions in various languages.

- REV_COLUMN_PROPERTIES – stores the column properties associated with each column.
- REV_TABLE_CLASS_ASSIGNMENT – stores the table classification associated with each table.

Note:

- In case custom instrument table contains the column in the same name as that of the super-class table, then column present in the custom instrument table will take precedence over the equivalent column of the super-class table. In case multiple super-class tables contain the same column, user should ensure that all the columns have same datatype, as any column can be selected and it is not resolved in any specific order.
- Physical order of the columns within the custom instrument table is determined in the following way:
  - Columns present in the custom instrument table.
  - Columns present in each of the linked super-class table. In case multiple super-class tables are linked to the custom instrument table, columns are rolled down from all super-class tables without any specific order.
- Within any table, ERwin maintains three different column orders:
  - Logical Order – Order of the columns as seen in Logical view of the model.
  - Physical Order – Order of the columns as seen in Physical view of the model.
  - Database Order – Order of the columns as seen in the Database schema.

Unique Index

Instrument tables require unique index on ID_NUMBER and IDENTITY_CODE column. This unique index needs to be created on the custom instrument table, post-model upload operation.

Transaction tables require unique index on ID_NUMBER, IDENTITY_CODE and one of the key dimension columns. This unique index needs to be created on the custom transaction table, post-model upload operation.

Note: The unique index may not contain any non-Key Dimension columns other than ID_NUMBER and IDENTITY_CODE..

Portfolio Selection

It enables users to define rules with a cross-instrument definition, since the Portfolio table classification contains columns (potentially including user-defined columns) that are common to all instruments. Any other specific instrument table selection (such as "Mortgages", and so on) would limit the rule definition to the specific instrument table. During processing of a "Portfolio" selection from an assumption rule, the engine will substitute the name of a specific table (e.g. instruments selected in the parent process rule).

Adding a new user defined column as a Portfolio column for use in all Instrument tables

Portfolio is available as a table selection in various modules including Infrastructure objects such as Filters and Expression rules. It is also available in application objects such as Profitability Management Allocation rules, and so on. To add a new user-defined column as a Portfolio column, use the following steps:

1. Include the column in the PORTFOLIO super-type table in the Erwin Data Model to ensure that the column rolls down to all subtype tables.

2. Complete incremental model upload to add the column to all subtype Portfolio tables.

3. Manually insert a row into the Atomic schema REV_PROPERTY_COLUMNS table with TABLE_PROPERTY_CD = 40:

   For example, if your new column is "APPLE_BRANCH_CD"

```
Insert into REV_PROPERTY_COLUMNS
(TABLE_PROPERTY_CD,COLUMN_NAME,PROTECTED_FLG) values
(40,'APPLE_BRANCH_CD',1);
COMMIT;
```

Object Registration Validation

Since leaf registration invalidates all Client Data Objects, Object Registration Validation procedure needs to be executed to validate the required tables. For more information on Object Registration Validation procedure, see Object Registration Validation, page 2-34.

Adding Custom Transaction Tables

Transaction tables are used within Profitability Management processing. There are seeded transaction tables that are packaged as part of Profitability Management

application. You can customize or remove any of them during implementation. In some cases, you might also require to add a custom transaction table.

The following topics are covered in this section:

- Super-class entities
- Steps in creating a custom transaction table
- Setting Table Classifications
- Setting Processing Key property
- Unique Index
- Object Registration Validation

Super-class Entities

Profitability Management processing mandates the transaction table to have a certain set of columns. These columns have been put together in super-class entities. The following are the seeded super-class entities:

- TRANS_LEAF_COLUMNS – contains the key dimension columns that are part of the Transaction tables.

Steps In Creating Custom Transaction Table

The following are the steps involved in creating a custom transaction table:

- Create a new subject area within the ERwin model.
- Move TRANS_LEAF_COLUMNS into the new subject area.
- Create the custom transaction table in ERwin. Specify logical name, physical name and description for the table. Define any columns that do not come from any of the standard super-class tables as part of the custom transaction table. Specify logical, physical names, domain and other column properties for each column.
- Create subtype relationship between the custom transaction table and TRANS_LEAF_COLUMNS super-class entity.

Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom transaction table:

- Choose Physical View within the ERwin model.
- Go to UDP tab within Table Properties window.
- Specify 'Yes' for 'Transaction Profitability' user defined property.

Once the model is prepared using the above steps, user should upload the ERwin model. After uploading the model, user can check if the custom transaction table has been created in the schema with columns from super-class entities that have been linked to the custom transaction table as well as the columns present in the custom transaction table. Model upload also creates metadata entries within the following Object Registration tables:

- REV_TABLES_B – Contains the list of table names.
- REV_TABLES_TL – contains the list of table display names and descriptions in various languages.
- REV_TAB_COLUMNS – contains the list of column names.
- REV_TAB_COLUMNS_MLS - contains the list of column display names and descriptions in various languages.
- REV_COLUMN_PROPERTIES - stores the column properties associated with each column.
- REV_TABLE_CLASS_ASSIGNMENT - stores the table classification associated with each table.

Note:

- In case custom transaction table contains the column in the same name as that of the super-class table, then column present in the custom transaction table will take precedence over the equivalent column of the super-class table.
- Physical order of the columns within the custom transaction table is determined in the following way:
  - Columns present in the custom transaction table.
  - Columns present in each of the linked super-class table.
- Within any table, ERwin maintains three different column orders:

- Logical Order – Order of the columns as seen in Logical view of the model.
- Physical Order – Order of the columns as seen in Physical view of the model.
- Database Order – Order of the columns as seen in the Database schema.

Setting Processing Key Property

'Processing Key' user defined property needs to be set for the following columns within the transaction table:

- ID_NUMBER
- IDENTITY_CODE
- Leaf columns that are part of the unique index

The following are the steps to set this property in ERwin:

- Choose Physical View within the ERwin model.
- Choose TRANS_LEAF_COLUMNS super-class table.
- Choose the leaf column that needs to be set 'Processing Key' property.
- Go to UDP tab in Column Properties window for this column.
- Specify 'Yes' against 'Processing Key' user-defined property.
- Choose the custom transaction table.
- Go to UDP tab in Column Properties window for ID_NUMBER and IDENTITY_CODE columns.
- Specify 'Yes' against 'Processing Key' user-defined property.

Unique Index

Transaction tables require unique index on the following columns:

- ID_NUMBER
- IDENTITY_CODE

- At-least one of the key dimension columns.

This unique index needs to be created on the custom transaction table, post-model upload operation.

Object Registration Validation

Since leaf registration in-validates all Client Data Objects, Object Registration Validation procedure needs to be executed to validate the required tables. For more information on Object Registration Validation procedure, see Object Registration Validation, page 2-34.

Adding Custom Lookup Tables

Lookup tables are used within OFSAA Profitability Management application. Lookup tables have to be created and registered within OFSAAI, in order to display them in Lookup Table Driver definition of OFSAA Profitability Management application.

The following topics are covered in this section:

- Steps in creating the lookup table in ERwin
- Setting Column Properties
- Setting Table Classifications
- Registering lookup tables and Validation
- Lookup Table Driver definition

Steps In Creating Lookup Table

Lookup table has to be created in the ERwin model. The following are the steps:

- Open the ERwin model in ERwin Data Modeler tool.
- Create a new subject area.
- Create a table and add columns to the table.
- Lookup table needs to at-least have one primary key column.
- Lookup table needs to at-least have one numeric non-key column. Such numeric columns will be the return value of the lookup.
- Specify logical names, comments and primary key for the table.
- Specify logical names, domains and comments for the column.

- Domains for the columns can be LEAF, BALANCE, RATE etc.
- Save the model.

Setting Column Properties

'Processing Key' is a column level User Defined Property (UDP) in ERwin model. This property can have two values – Yes or No. 'Processing Key' property needs to be set for all the primary key columns of the lookup table.

'Balance Range' is a column level User Defined Property (UDP) in ERwin model. This property can have two values – Yes or No. 'Balance Range' property needs to be set for the columns that can have range values in the lookup.

The following are the steps for setting the above properties:

- Open the ERwin model in ERwin Data Modeler tool.
- Go to the subject area where lookup table was created.
- Choose the table and open the columns of the table.
- Go to UDP tab within the column properties for each column.
- Specify the value for the required user defined properties.
- Save the model.

Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom lookup table:

- Choose Physical View within the ERwin model.
- Go to UDP tab within Table Properties window.
- Specify 'Yes' for 'PA Lookup Tables' user-defined property.

Registering Lookup Tables and Validation

Upload the model and execute the object registration validation.

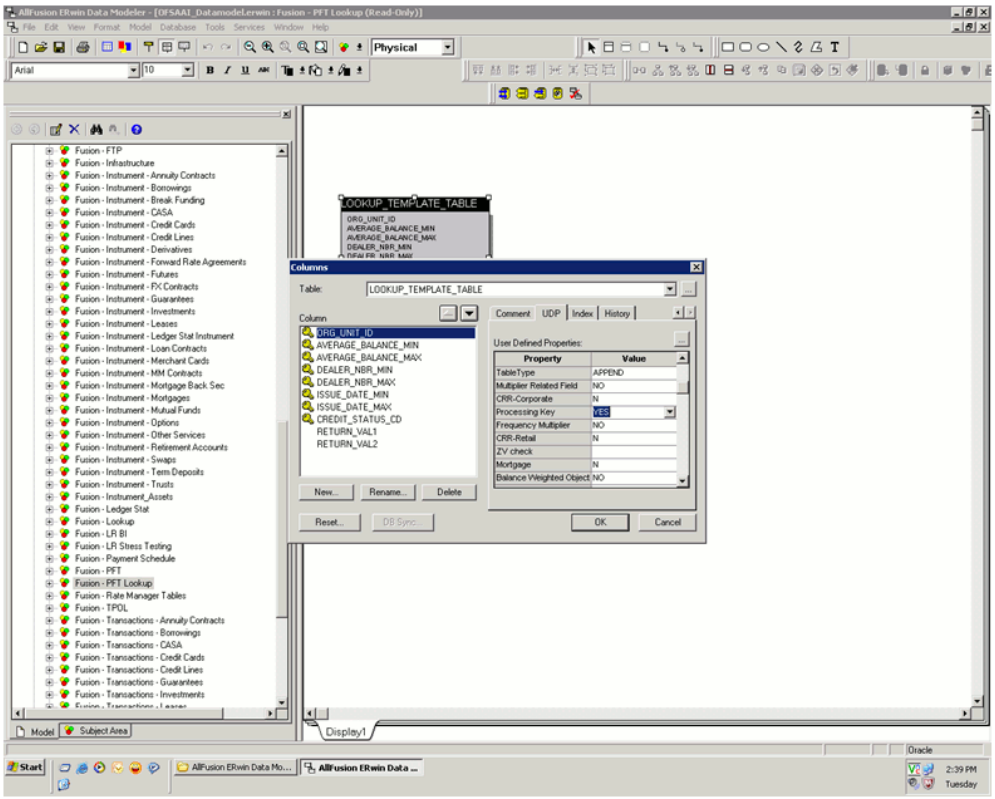
Lookup Table Driver Definition

Post registration and validation, the lookup table is available within Lookup Table Driver definition of OFSAA Profitability Management application.

Following is the criteria for columns to be displayed in the Source - Lookup Mapping grid:

- Column needs to be Primary Key or be part of composite primary key.
- 'Processing Key' user defined property should be set for the column under UDP tab as shown below.

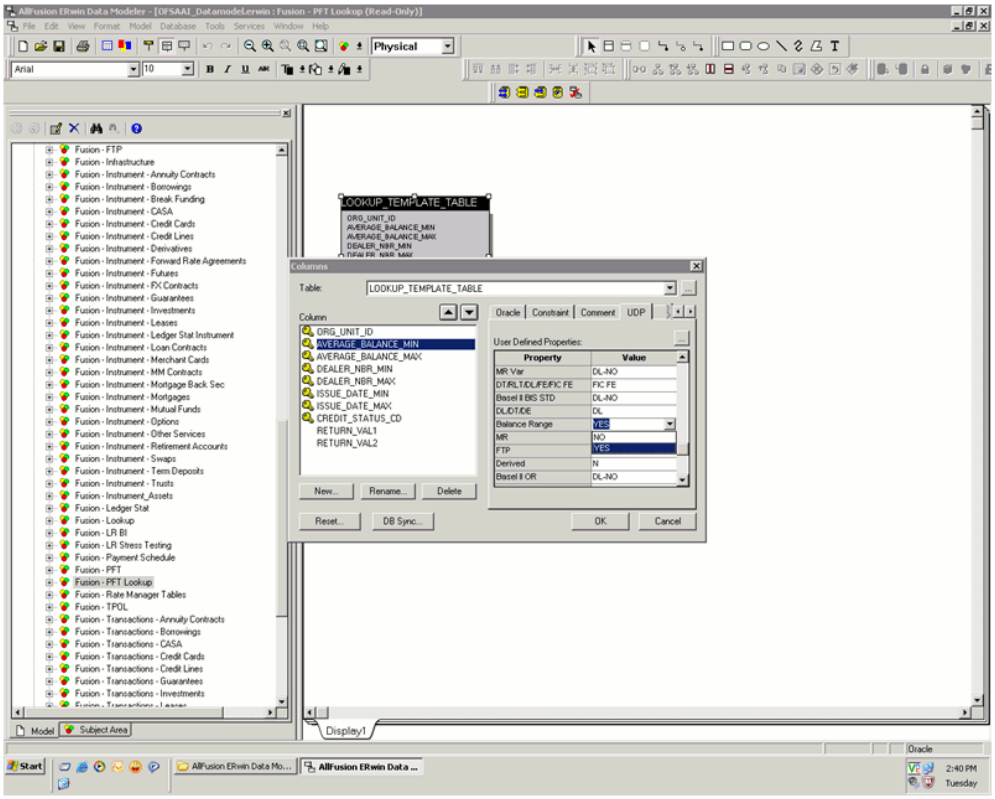
Mapping of Column to Processing Key



Following is the criteria for columns to be part of the Range:

- 'Balance Range' user defined property should be set for the column under UDP tab as shown below.

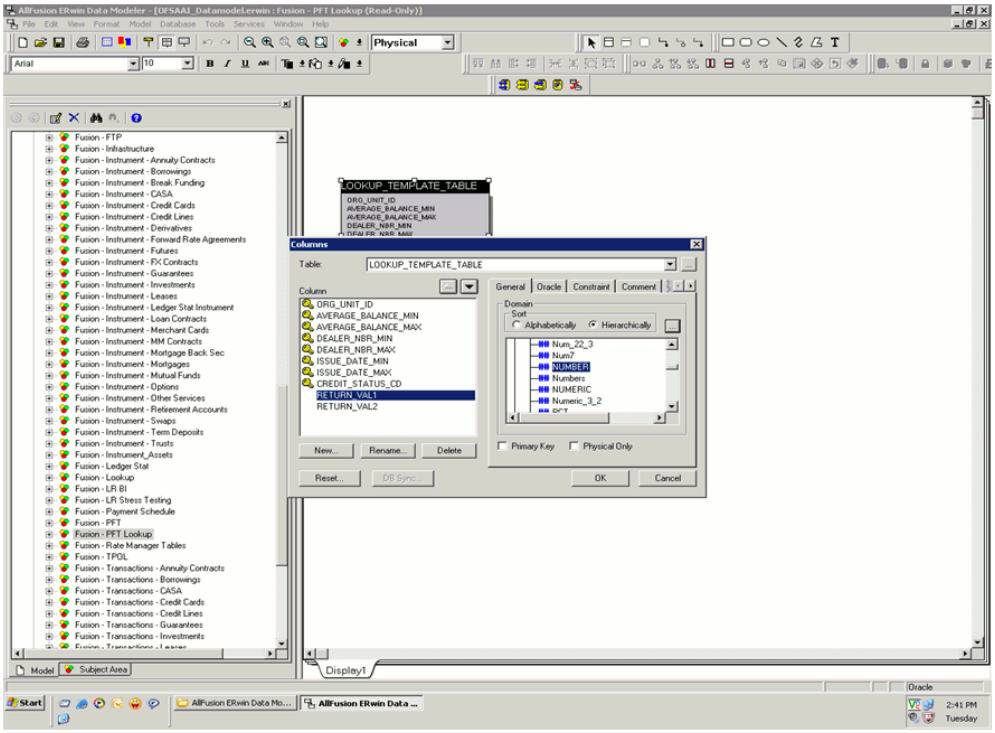
Mapping of Column for Range Property



Following is the criteria for columns to be part Lookup Return Value:

- Column should not be primary key/processing key or be part of composite primary key.
- Column domain should be defined as NUMBER under General Tab as shown below.

Mapping of Column for Look up Return Value



Object Registration And Validation

Table Classifications provide a means to designate how tables are used within the OFSAA suite of applications. Each table classification identifies a specific purpose for which an assigned table is allowed to be used.

Some Table Classifications have requirements that must be satisfied in order for an object to be assigned to the classification. These requirements are designated by Table Properties associated to the Table Classifications. These Table Properties are either specific column name requirements or logic validations.

Table Classification assignments are stored in REV_TABLE_CLASS_ ASSIGNMENT.

Object Registration is a process of classifying a table with one or more table classifications depending on the purpose of the table. This step is performed within the ERwin model by setting various User Defined Properties for a client data object. Validation procedure validates table class assignment for a client data object and needs to be executed after model upload operation.

The following topics are covered in this section:

- User-Assignable Table Classifications

- Requirements for each Table Classification
- Validation procedure
- Executing the Validation Procedure
- Exception Messages

User-Assignable Table Classification

User-Assignable Table Classifications are those that can be assigned by the administrator to user-defined and client data objects, including the OFSAAI Instrument tables. These Table Classifications identify processing and reporting functions for the OFSAA. Some of these Table Classifications have requirements that must be met in order for the classification to be assigned to a table or view.

All User-Assignable Table Classifications are available for assignment within the ERwin model. The following table lists the User-Assignable Table Classifications:

Code	Table Classification Name
20	Instrument
50	Ledger Stat
100	Portfolio
200	TP Cash Flow
210	TP Non-Cash Flow
295	Codes User Defined (base tbl)
296	MLS Descriptions User Defined
300	Transaction Profitability
310	Instrument Profitability
320	User Defined
330	Data Correction Processing
360	RM Standard

Code	Table Classification Name
370	TP Option Costing
500	PA Lookup Tables
600	Derivative Instruments
530	Break Funding
197	MLS Descriptions Reserved
198	Codes Reserved (base tbl)

Requirement For Table Classification

OFSAAI requires specific table structures, column names and column characteristics for OFSAA operations. These structures and requirements are embodied by the User-Assignable Table Classifications.

Each Table Classification comprises individual Table Properties that define the requirements for that classification. Table Properties are two distinct types: those encompassing specific column requirements and those encompassing logic requirements via stored procedures.

The following table provides the validation checks that are being done for each of the table classification:

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
50	Ledger Stat	Ledger Leaf Column Class	Fields that are part of core modeling dimensions for Fusion PFT	Checks if columns of super-type Ledger Leaf Column Class is present

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
100	Portfolio	Portfolio Requirements	Dynamic list of Portfolio fields	Checks if columns of super-type Portfolio Requirements is present
200	TP Cash Flow	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
200	TP Cash Flow	Cash Flow Proc. Requirements	Fields required by TP and RM Cash Flow processing	Checks if columns of super-type Cash Flow Proc. Requirements is present
200	TP Cash Flow	Cash Flow Edit Requirements	Fields required by Cash Flow Edits in addition to Cash Flow fields	Checks if columns of super-type Cash Flow Edit Requirements is present
200	TP Cash Flow	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
200	TP Cash Flow	TP Basic Requirements	Non-cash flow Transfer Pricing fields	Checks if columns of super-type TP Basic Requirements is present

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
200	TP Cash Flow	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimesion leaf columns. The leaf columns should be of data type NUMBER
200	TP Cash Flow	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, RM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
210	TP Non-Cash Flow	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
210	TP Non-Cash Flow	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
210	TP Non-Cash Flow	TP Basic Requirements	Non-cash flow Transfer Pricing fields	Checks if columns of super-type TP Basic Requirements is present
210	TP Non-Cash Flow	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimesion leaf columns. The leaf columns should be of data type NUMBER

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
210	TP Non-Cash Flow	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, RM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
300	Transaction Profitability	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
300	Transaction Profitability	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
300	Transaction Profitability	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimesion leaf columns. The leaf columns should be of data type NUMBER
300	Transaction Profitability	Validate Transaction Key	Validate the unique key for Transaction Profitability tables	Transaction table should have composite index present on ID_NUMBER and IDENTITY_CODE and all the processing key columns.

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
310	Instrument Profitability	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
310	Instrument Profitability	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
310	Instrument Profitability	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimesion leaf columns. The leaf columns should be of data type NUMBER
310	Instrument Profitability	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, RM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
330	Data Correction Processing	Validate Processing Key	Validate the unique key for Processing tables	Processing Key Column for a table have a matching unique index
360	ALM Standard	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
360	ALM Standard	Cash Flow Proc. Requirements	Fields required by TP and RM Cash Flow processing	Checks if columns of super-type Cash Flow Proc. Requirements is present
360	ALM Standard	Cash Flow Edit Requirements	Fields required by Cash Flow Edits in addition to Cash Flow fields	Checks if columns of super-type Cash Flow Edit Requirements is present
360	ALM Standard	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
360	ALM Standard	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimesion leaf columns. The leaf columns should be of data type NUMBER
360	ALM Standard	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, RM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
370	TP Option Costing	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
370	TP Option Costing	Cash Flow Edit Requirements	Fields required by Cash Flow Edits in addition to Cash Flow fields	Checks if columns of super-type Cash Flow Edit Requirements is present
370	TP Option Costing	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
370	TP Option Costing	TP Option Costing Requirements	Fields required for Transfer Pricing Option Costing processing	Checks if columns of super-type TP Option Costing Requirements is present
370	TP Option Costing	TP Basic Requirements	Non-cash flow Transfer Pricing fields	Checks if columns of super-type TP Basic Requirements is present
370	TP Option Costing	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimesion leaf columns. The leaf columns should be of data type NUMBER
370	TP Option Costing	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, RM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
500	PA Lookup Tables	Validate PA Lookup	Procedure to check if there is a primary key for the lookup tables.	Validation. All Lookup table should have a primary key present
530	Break Funding	Break Funding Requirements	Fields required as part of TP break funding	Checks if columns of super-type Break Funding Requirements is present

Specific column requirements for each table property can be obtained by querying REV_COLUMN_REQUIREMENTS table.

Validation Procedure

The OFSA_TAB_CLASS_REQ package contains all of the procedures and supporting functions that validates if a table meets the requirements for a particular Table Classification.

The package performs the following validations:

- VALIDATE_INST_KEY

  This procedure validates if a table has ID_NUMBER and IDENTITY_CODE, or ID_NUMBER, IDENTITY_CODE and AS_OF_DATE as its unique index and if the Processing key designated in Column Properties is ID_NUMBER, IDENTITY_CODE.

- UPDATABLE_INST_REQ_FIELDS

  This procedure checks that all of the Instrument Required Fields are also listed as updatable in USER_UPDATABLE_COLUMNS for the specified table or view.

- VALIDATE_INST_LEAVES

  This procedure will validate a table has all the required leaf columns

- VALIDATE_TRANS_KEY

  This procedure validates if a table has ID_NUMBER and IDENTITY_CODE and one

or more 'B' Leaf Columns in its unique index and that these columns match the Processing key designated in Column Properties.

- VALIDATE_CORR_KEY

  This procedure will validate a table has a unique index with updatable columns.

All the above procedures return a success or failure status. The REV_TAB_CLASS_ASSIGNMENT table is updated as 'Y' if a table is successfully validated and 'N' in case of failure.

Executing the Validation Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The syntax for calling the procedure is:

```
Declare
Result number;
begin
     result := fsi_batchtableclassreq(pbatchid, pmis_date);
end;
```

An example of running the stored procedure from SQL*Plus

```
SQL> var output number;
SQL> exec fsi_batchtableclassreq('VALIDATE_DATAMODEL' , '20100809');
```

> **Note:** Since the package contains huge number of dbms_output statements, user should either increase the output buffer size or disable the server output.

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- *Batch_Table_Class_Req*
- Parameter List:- Batch Identifier and MISDATE

To execute the same procedure in SQL*Plus/PlSQL Developer/SQL PLUS:

```
set serveroutput off;

begin
    ofsa_tab_class_req.validate_all_tab_class('1236','25-JAN-2010');
end;
```

Exception Messages

The OFSA_TAB_CLASS_REQ packages throws the following exceptions.

Exception 1: FAILED: Table Property 1030 - Validate Correction Key

This exception occurs when no valid unique index found.

Exception 2: FAILED: Table Property 1030 - Validate Correction Key

This exception occurs when Processing Key Column Properties do not match unique index

Exception 3: FAILED: Table Property 1030 - Validate Transaction Key

This exception occurs when no valid unique index found.

Exception 4: FAILED: Table Property 1000 - Validate Instrument Leaves

This exception occurs when one or more Leaf Columns are missing or incorrectly registered. Check if the datatype of the LEAF columns is NUMBER and domain of these columns is LEAF.

Defining Alternate Rate Output Columns

This section details the steps required for defining Alternate Rate Output columns within the OFSAA Fund Transfer Pricing Application.

The following topics are covered in this section:

- Setting User Defined Properties in ERwin
- Uploading the model and object registration

User-Defined Properties

The following are the user-defined properties that are available for identifying columns required for alternate rate output:

- Transfer Pricing Output (Column Property – 80)
- Option Cost Output (Column Property – 81)

- Other Adj Spread Output (Column Property – 82)
- Other Adj Amount Output (Column Property – 83)

User needs to assign one of the above properties to the columns that need to be used as Alternate Rate Output columns within the Fund Transfer Pricing application.

The following are the steps to set the user-defined property to the column:

- Open the ERwin file in ERwin Data Modeler tool.
- Go to Main Subject Area.
- Go to Physical View.
- Choose the entity that contains the alternate rate output column. This entity can also be a super-type (like TP_BASIC_REQ).
- Select the column and open the column properties for the column.
- Go to UDP tab within column properties.
- Select 'YES' for one of the above user-defined properties.
- Save the model.

> **Note:** Setting the user-defined property of the columns within a super-type entity will apply to all the entities that are related to the super-type.

Uploading the Model

Upload the model in OFSAAI and perform object registration. After uploading the model, user can execute the below query to check if the user-defined properties are set for the columns.

```
select * from rev_column_properties where column_property_cd in
(80,81,82,83)
where TABLE_NAME = <<table_name>>
```

Replace <<table_name>> with the relevant table name and column name in the above query and execute the same. Above query returns the columns that are used for alternate rate outputs.

User Defined Properties

User Defined Properties are set for tables and columns within ERwin.

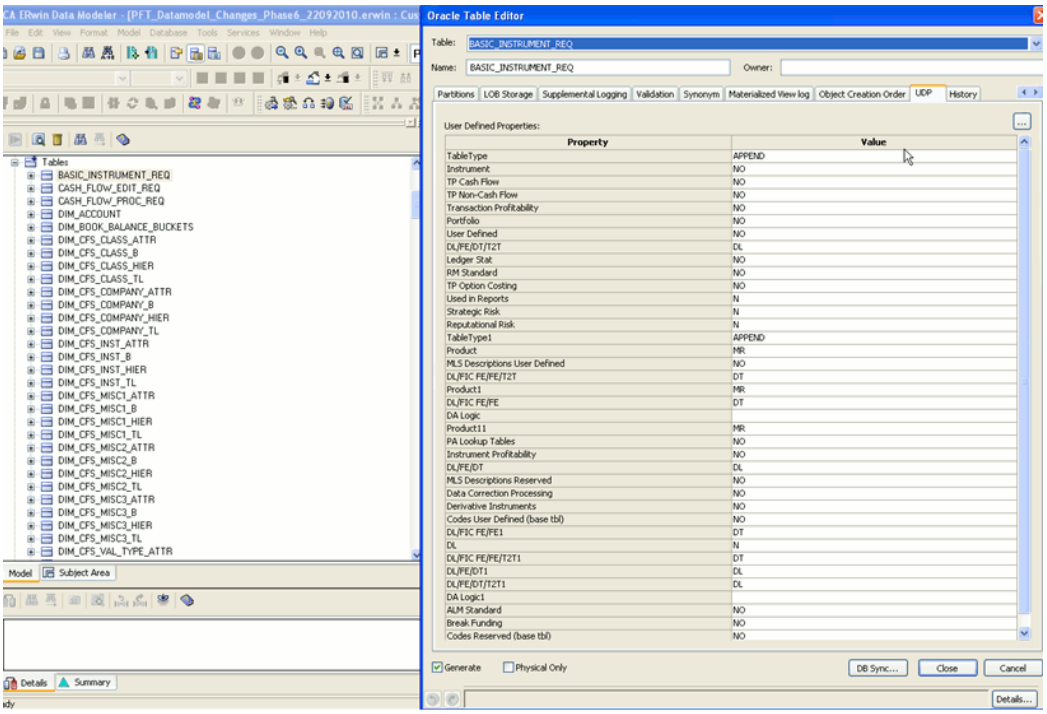


Table Level User Defined Properties

The following user defined properties can be set for the table:

UDP Name	Description	List of values
Instrument	Property to identify if the table is classified as a basic instrument table. (that is, Instrument table classification code 20)	YES / NO
TP Cash Flow	Property to identify if the table is classified as 'TP Cash Flow' for the purpose of generating Transfer Pricing rates using cash flow methods.	YES / NO
TP Non Cash Flow	Property to identify if the table is classified as 'TP Non-Cash Flow' for the purpose of generating Transfer Pricing rates using non cash flow methods.	YES / NO
Transaction Profitability	Property to identify if the table is classified as 'Transaction' for the purpose of executing allocation rules.	YES / NO

UDP Name	Description	List of values
Portfolio	Property to identify if the table is classified as 'Portfolio'.	YES / NO
User Defined	Property to identify if the table is classified as 'User Defined' table for storing multi-lingual descriptions for codes.	YES / NO
Ledger Stat	Property to identify if the table is classified as 'Ledger Stat' for the purpose of executing allocation rules.	YES / NO
ALM Standard	Property to identify if the table is classified as 'ALM Standard' for the purpose of executing ALM cash flow engine to generate cash flows.	YES / NO
TP Option Costing	Property to identify if the table is classified as 'TP Option Costing' for the purpose of generating Transfer Pricing rates with option costing.	YES / NO
Break Funding	Property to identify if the table is classified as 'Break Funding' for the purpose of generating Break funding charges using Transfer Pricing engine.	YES / NO
MLS Descriptions Reserved	Property to identify if the table is classified as 'Reserved' table for storing multi-lingual descriptions for codes.	YES / NO
Codes Reserved (base tbl)	Property to identify if the table is classified as 'Reserved' table for storing codes of simple dimensions.	YES / NO
Codes User Defined (base tbl)	Property to identify if the table is classified as 'User-defined' table for storing codes of simple dimensions.	YES / NO
PA Lookup Tables	Property to identify if the table is classified as 'Lookup Table' for the purpose of defining lookup table allocation rules.	YES / NO
Instrument Profitability	Property to identify if the table is classified as 'Instrument' for the purpose of executing allocation rules.	YES / NO

UDP Name	Description	List of values
Derivative Instruments	Property to identify if the table is classified as 'Derivatives' for the purpose of executing ALM cash flow engine to generate cash flows for derivative instruments.	YES / NO
Data Correction Processing	Property to identify if the table is classified as 'Data Correction Processing' for the purpose of executing Cash Flow Edits engine.	YES / NO

Column Level User Defined Properties

The following user defined properties can be set for the column:

UDP Name	Description	List of values
Balance Range	Property to identify if the column within a table classified as 'PA Lookup Table' must be displayed under 'Range' within Lookup table definition.	YES / NO
Balance	Property to identify if the column is of type 'Balance'.	YES / NO
Standard Rate	Property to identify if the column is of type 'Standard Rate'.	YES / NO
Balance Weighted Object	Property to identify if the column is of type 'Balance Weighted Object'.	YES / NO
Processing Key	Property to identify if this column is used as a 'Processing Key' within the instrument, transaction and ledger_stat table.	YES / NO
Frequency Multiplier	Property to identify if the column is used to store 'Frequency'. This property is used in Filters UI within OFSAAI.	YES / NO
Multiplier Related Field	Property to specify the name of the column that is used to store the multiplier for the corresponding 'Frequency' column. This property is used in Filters UI within OFSAAI.	Text

UDP Name	Description	List of values
Related Field	Property to specify the name of the column that is used to store the multiplier for the corresponding 'Term' column. This property is used in Filters UI within OFSAAI.	Text
Term Multiplier	Property to identify if the column is used to store 'Term'. This property is used in Filters UI within OFSAAI.	YES / NO
Column Alias	Property to specify an alias for the column. This is used within the staging loader program for loading LEDGER_STAT table.	Text
Statistic	Property to identify if the column is of type 'Statistic'.	YES / NO
Transfer Pricing Output	Property to identify if the column must be set as an alternate output column for writing transfer rates by transfer pricing engine.	YES / NO
Option Cost Output	Property to identify if the column must be set as an alternate output column for writing option costing output by transfer pricing engine.	YES / NO
Other Adj Spread Output	Property to identify if the column must be set as an alternate output column for writing other adjustment spread by transfer pricing engine.	YES / NO
Other Adj Amount Output	Property to identify if the column must be set as an alternate output column for writing other adjustment amount by transfer pricing engine.	YES / NO

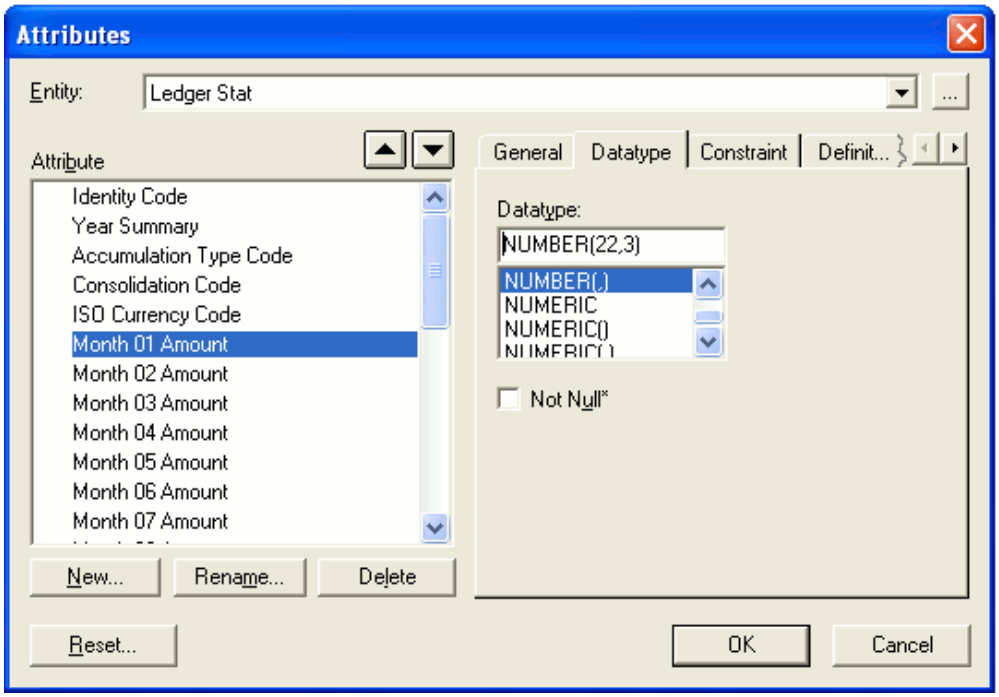
Modifying the Precision of Balance Columns In Ledger Stat

Steps to modify the Precision

1. Open the ALM/FTP/PFT model using AllFusion ERwin Data Modeler.
2. Switch to **Fusion – Ledger Stat** subject area.
3. Select **Logical** view.
4. Edit the Ledger Stat table by double clicking the table in the Logical Layer.

5. Change the data type in **Datatype** tab to the revised precision and scale (example, NUMBER (22, 3)) for the following columns:

   - Month 01 Amount, Month 02 Amount, Month 03 Amount and so on.
   - YTD 01 Amount, YTD 02 Amount, YTD 03 Amount and so on.

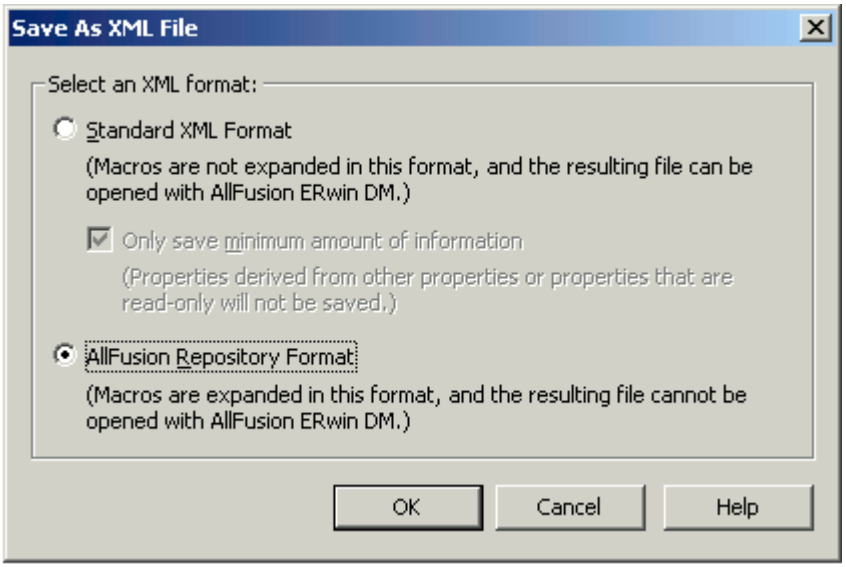


6. Save the changes.

7. Select the **Physical** view.

8. Click LEDGER_STAT table and view the datatype of columns – MONTH_01 till MONTH_12 and YTD_01 till YTD_12. The data type of these columns should display the new precision and scale.

9. Save the model as xml in **AllFusion Repository Format**.



10. Perform incremental model upload.

    Note: In case, users decrease the precision and scale for the columns, such columns should not have any values during model upload.

3

Utilities

This chapter details the steps involved in executing various data model utilities that are available within OFSAA.

This chapter covers the following topics:

- Reverse Population
- Product Instrument Mapping
- Instrument Synchronization
- Stage Synchronization
- Ledger Load Undo

Reverse Population

Reverse population procedure populates dimension members, attributes and hierarchies from new dimension tables to OFSA legacy set of dimension tables. ALM, TP and PFT engines refer to OFSA legacy tables for retrieving dimension member information.

The following topics are covered in this section:

- Tables that are part of Reverse Population
- Reverse Population procedure
- Executing the Reverse Population Procedure
- Exception Messages

Tables As Part Of Reverse Population

Dimension data is stored in the following set of tables:

- DIM_<DIMENSION>_B - Stores leaf and node member codes within the dimension.
- DIM_<DIMENSION>_TL - Stores names of leaf and node and their translations.
- DIM_<DIMENSION>_ATTR - Stores attribute values for the attributes of the dimension.
- DIM_<DIMENSION>_HIER - Stores parent-child relationship of members and nodes that are part of hierarchies.

Data present in the above set of dimension tables are transformed into the below set of OFSA Legacy tables.

The reverse population routine synchronizes the dimension data between the new dimension tables and the OFSA Legacy tables. Reverse population occurs automatically if enabled in the AMHMConfig.properties file. In the AMHMConfig.properties file, set the Parameter value to Y for a specific Dimension Id. The setting in the AMHMConfig.properties only impacts dimension values entered through the interface. Reverse population must be executed as a batch for bulk loading. For more information on how to define the reverse populate parameters in the AMHMConfig.properties file, see *Oracle Financial Services Analytical Applications Infrastructure (OFSAAI) Installation and Configuration Guide.*

- OFSA_LEAF_DESC – Stores the description of leaf members that are part of the dimension.
- OFSA_NODE_DESC – Stores the description of nodes that are used within the hierarchy.
- OFSA_DETAIL_LEAVES – Stores the attributes of Common COA dimension.
- OFSA_DETAIL_OTHER_COA – Stores the attributes of GL or Product or any other key dimension.
- OFSA_DETAIL_ELEM_B/OFSA_DETAIL_ELEM_MLS – Stores the attributes of Financial Elements dimension.
- OFSA_IDT_ROLLUP – Stores the hierarchy as level-based.
- OFSA_LEVEL_DESC – Stores the hierarchy levels.

Reverse population is done for all key dimensions that are configured within the OFSAAI framework.

Reverse Population Procedure

The REVERSE_POPULATION package populates the OFSA legacy dimension tables from new dimension tables.

The procedure performs the following functions:

- Gets the list of source and target tables. The source tables for given dimension is stored in REV_DIMENSION_B table. The OFSA target table for a given dimension is stored in OFSA_CATALOG_OF_LEAVES.
- The REVERSE POPULATION transposes the seeded attributes, leaf members and hierarchy data stored in the form of rows (new dimension table structure) to columns (OFSA).
- All exception messages are logged in the FSI_MESSAGE_LOG table.

After the Reverse Population procedure is completed, you should query the OFSA legacy tables to look for dimension members.

Executing the Reverse Population Function

You can execute this function from either within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the function with a PL/SQL block, follow the below steps:

- **Members Reverse Population**

```
Function fsi_batchMemberLoad(batch_run_id   varchar2,
                             mis_date       varchar2,
                             pDimensionId   varchar2,
                             pMemberId      varchar2,
                             pMode          varchar2)
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- MIS_DATE in the format YYYYMMDD.
- pDIMENSIONID is the dimension id.
- pMEMBERID. This can be null. If value is provided, only that member id gets reverse populated.
- If pMode value is 1, it means fresh insert, if value is 2 means update, and if value is 3 means delete. In batch mode, you can prefer to use 2.

For Example:

```
Declare
    num number;
Begin
    num := fsi_batchmemberload
('INFODOM_20100405','20100405',1,null,2);
End;
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- *Batch_Member_Load*
- Parameter List:- Dimension ID, Member id

- **Hierarchy Reverse Population**

```
Function fsi_batchhierarchyload(batch_run_id   varchar2,
                                mis_date        varchar2,
                                pDimensionId    varchar2,
                                pHierarchyId    varchar2,
                                pMode           varchar2)
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- MIS_DATE in the format YYYYMMDD.
- pDIMENSIONID is the dimension id.
- pHIERARCHYID. This can be null. If value is provided, only that Hierarchy gets reverse populated.
- If pMode value is 1, it means fresh insert, if value is 2 means update, and if value is 3 means delete. In batch mode, you can prefer to use 2.

For Example:

```
Declare
    num number;
Begin
    num := fsi_batchhierarchyload('INFODOM_20100405','20100405'
,1,null,2);
End;
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list

- Rule Name:-*Batch_Hier_Load*
- Parameter List:- Dimension ID, Hierarchy id

> **Note:** The reverse population fsi_batchMemberLoad and fsi_batchHierarchyLoad should be executed after fn_drmdataloader. The fsi_batchMemberLoad reverse populates the members and the fsi_batchHierarchyLoad reverse populates the hierarchies to the legacy structures.

Exception Messages

The Reverse Population procedure may cause some exceptions to appear. The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle them so that your program can proceed.

Exception 1: Error. While getting dimension details

This exception occurs when the reverse population procedure cannot find any data configured in the driver table (REV_DIMENSIONS_B).

Exception 2: Error. While generating hierarchy Query

This exception occurs when there is a problem generating hierarchy query dynamically.

Exception 3: Error. While populating Nodes

This exception occurs when there is an error populating the OFSA_NODE_DESC table.

Product Instrument Mapping

ALM and TP processes can be based on a set of data tables or a set of products. In case products are selected, ALM and TP engine internally gets the list of data tables mapped to these products and processes those data tables. During the period-ending load cycle, data is loaded into Client Data Objects such as Instrument tables. During this load process, all the distinct members of 'Product' type dimension that are present within each data table will be stored in a separate table (FSI_M_PROD_INST_TABLE_MAP) by executing Product Instrument mapping procedure.

The following topics are covered in this section:

- Tables requiring Product-Instrument table map
- Product-Instrument table map procedure
- Executing the Product-Instrument table map Procedure

- Exception Messages

Tables Requiring Synchronization

Product-instrument table mapping is required only for Instrument tables. Instrument tables are defined as all tables with the *Instrument* Table Classification (table_classification_cd in (20,600,200,210)) on which all of the defined Leaf Columns exist.

Product Instrument Table Map Procedure

This function gives exact mapping of a particular 'Product' stored in multiple Instrument table, and mapping is stored in FSI_M_PROD_INST_TABLE_MAP for given AS_OF_DATE. The function outputs the mapping information only if the corresponding 'Product' definition exits in the corresponding dimension table.

The procedure performs the following functions:

- Gets the list of 'Product' type dimensions from dimension registry table (REV_DIMENSIONS_B).
- Gets the list of Instrument tables from REV_TABLE_CLASS_ASSIGNMENT.
- Fetches the distinct set of members for each 'Product' type dimension from all instrument tables for a given AS_OF_DATE.
- Stores the above set into a mapping table (FSI_M_PROD_INST_TABLE_MAP).
- The function outputs message in the message log if the member definition which exists in the Instrument table is not found in the respective dimension table.

After the Product-Instrument table mapping utility run is completed, you should query the mapping table to look for dimension members that are present as part of each instrument table.

Executing the PRODUCT_INSTRUMENT_TABLE_MAP Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The procedure requires 3 parameters – Batch Id – which can be used to see the log of the procedure executed, MISDATE and the AS_OF_DATE. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples. The syntax for calling the procedure is:

```
Declare
output number;
Begin
  Output:= fn_Product_Instrument_Map ('Batch_Id',
'MISDATE','AS_OF_DATE');
End;
```

- AS_OF_DATE is the date for which mapping is required.
- MISDATE is the date for which batch is run.

Both MISDATE and AS_OF_DATE should be passed as 'YYYYMMDD' format.

An example of running the function from SQL*Plus for the FSI_D_TERM_DEPOSITS table follows:

```
SQL>  var output number;
SQL> execute  :output:= fn_Product_Instrument_Map ('Batch_Id',
'20100131,'19991231');
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows. Call the procedure as often as required to synchronize all of your instrument tables. The appropriate table parameters are enclosed in single quotes.

```
SQL> declare
     output number;
     begin
     output:=  fn_Product_Instrument_Map ('Batch_Id',
'MISDATE','AS_OF_DATE')
     end;
     /
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type :- Select appropriate datastore from list
- Datastore Name :- Select appropriate name from the list
- IP address :- Select the IP address from the list
- Rule Name :- *Product_Inst_Mapping*
- Parameter List :- AS_OF_DATE

> **Note:** BATCHID and MISDATE will be passed explicitly in ICC framework.

Exception Messages

The Product to Instrument Mapping function may cause two exceptions to appear. The text and explanation for each of these exceptions follows. If you call the function from a PL/SQL block you may want to handle them so that your program can proceed.

Exception 1: Table does not exist

The exception message reads:

```
Table 'TABLE_NAME' does not exist.
```

This exception occurs when the function does not find the Instrument table.

Exception 2: Column does not exist

The exception message reads:

```
Column 'Column_Name' does not exists in the instrument table
'Table_Name' while processing dimension 'Dimension ID'.
```

This error occurs when leaf column does not exist in the Instrument table.

Instrument Synchronization

During the period-ending load cycle, data is loaded into Client Data Objects such as Instrument tables and the LEDGER_STAT table. During this load process, it is possible for new, unidentified Dimension and Code values to be loaded into these tables.

The Instrument Synchronization procedure identifies these new Dimension and Code values and inserts default description entries for them into the appropriate tables. The procedure performs both of these synchronizations simultaneously. OFSAAI requires that all Dimension and Code values have a corresponding description. This is required for any OFSAA reporting operation to return the correct results. It also ensures that Hierarchies work properly within the OFS analytical applications.

The following topics are covered in this section:

- Tables Requiring Synchronization
- Dimension Member Synchronization
- Code Synchronization
- Executing the Synchronize Instrument Procedure
- Exception Messages

Tables Requiring Synchronization

Dimension member and Code value synchronization is required only for Instrument and LEDGER_STAT tables. Instrument tables are defined as all tables with the *Instrument* Table Classification (table_classification_cd = 20) on which all of the defined Key Dimension Columns exist.

Dimension Member Synchronization

The SYNCHRONIZE_INSTRUMENT procedure synchronizes the dimension member tables and the hierarchy tables with LEDGER_STAT and instrument tables, using default values for member descriptions and other information columns. You can then add the correct data to the new dimension members in AMHM member maintenance.

The procedure performs the following functions:

- Checks the specified table (LEDGER_STAT or instrument) for new dimension members in each of that table's key dimension columns and adds the new dimension value as leaf members to the respective dimension member tables.
- Adds the new dimension member to the corresponding attribute tables with default values for mandatory attributes.
- When new dimension members are added to the dimension tables these members include *'No Description'* in the DESCRIPTION column and contain default values for mandatory attributes.
- Reverse populates the newly added dimension members into legacy OFSA tables. During reverse population, new members are created as orphan members, under corresponding hierarchies.

After the SYNCHRONIZE_INSTRUMENT utility run is completed you should look for any new dimension members using the AMHM member maintenance UI and enter the correct descriptions and other member information. You should also look at the orphan node of each Hierarchy for new dimension members and move these members to the appropriate branch in the rollup.

Codes Synchronization

The SYNCHRONIZE_INSTRUMENT procedure identifies code values in Instrument and LEDGER_STAT tables for which a corresponding description does not exist and inserts a default description into the appropriate Code Description object. This applies only to CODE columns categorized as User-Editable or User-Defined (refer table classification). CODE columns for which OFSAA reserves all of the values are not updated by this procedure. The procedure displays a warning message for any unidentified values in CODE columns where OFSAA reserves the entire range.

For each CODE column (REV_DATA_TYPE_CD equals 3) on the specified object, the SYNCHRONIZE_INSTRUMENT procedure queries from REV_DESCRIPTION_TABLES to identify the object storing the corresponding descriptions. If the resulting object is a User-Editable or User-Defined Code Description object (checks from REV_TABLE_CLASS_ASSIGNMENT table), then the procedure inserts a default description for any code values for which a description record does not already exist. If the resulting object is an OFSAA Reserved Code Description object, then the procedure outputs a warning message indicating how many invalid code

values exist in the specified Instrument or LEDGER_STAT table **in the message log (FSI_MESSAGE_LOG)**.

For example, if you are synchronizing the FSI_D_TERM_DEPOSITS table, the procedure queries all of the CODE columns on this table. An example of a Reserved CODE column is ACCRUAL_BASIS_CD. If the procedure finds any code values in this column that are not present in the corresponding Code Description object (FSI_ACCRUAL_BASIS_CD), it outputs an error message indicating the number of invalid values present. OFSAA Reserved Code Description objects are identified by the following SQL statement:

```
select table_name from rev_table_class_assignment
where table_classification_cd = 197;
```

An example of a User-Editable CODE column is SIC_CD. If the procedure finds any code values in SIC_CD in the FSI_D_TERM_DEPOSITS table that do not have a description in FSI_SIC_MLS, it creates a default description 'No Description' for each value. It is then up to the users to update these descriptions as appropriate. User-Editable Code Description objects are identified by the following SQL statement:

```
select *  from rev_description_tables
          where table_name = 'FSI_D_TERM_DEPOSITS'
           and description_table_name not in
          (select table_name   from rev_table_class_assignment
              where table_classification_cd = 197)
```

Executing the SYNCHRONIZE_INSTRUMENT Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The procedure requires 2 parameters - table name to be synchronized and the As of Date. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples. The syntax for calling the procedure is:

```
Declare
 output number;
Begin
   synchronize_instrument('Batch_Id', 'TABLE_NAME', output)
End;
```

where table_name is either:

- The name of an Instrument table
- LEDGER_STAT

An example of running the stored procedure from SQL*Plus for the FSI_D_TERM_DEPOSITS table follows:

```
SQL>  var output number;
SQL>
synchronize_instrument('INFODOM_20101231','FSI_D_TERM_DEPOSITS',:output)
;
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows. Call the procedure as often as required to synchronize all of your instrument tables. The appropriate table name and AS_OF_DATE is enclosed in single quotes.

```
SQL> declare
     output number;
     begin
       synchronize_instrument('INFODOM_20101231','LEDGER_STAT',output);
     end;
     /
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- fn_Synchronize_Instrmts
- Parameter List:- Instrument Table Name or LEDGER_STAT

Exception Messages

The SYNCHRONIZE_INSTRUMENT procedure may cause some exceptions to appear. The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle them so that your program can proceed.

Exception 1: Table is not an Instrument or LEDGER_STAT table

The exception message reads:

```
ORA-20002 Cannot process: table_name is not an OFSA Instrument or Ledger
type table having all leaf columns.
```

This exception occurs when the table_name parameter is not designated as an Instrument table or LEDGER_STAT table in the OFSAA Metadata. The procedure identified such tables based upon the Table Classification (Instrument or LEDGER_STAT).

Exception 2: Table has invalid seeded FINANCIAL_ELEM_ID values

The exception message reads:

```
ORA-20004 Cannot process: table_name has new FINANCIAL_ELEM_ID values
that are within seeded range (less than 10000).
```

This error occurs when user-defined leaf values are found in the DIM_FINANCIAL_ELEMENTS_B table within the FDM Reserved seeded data range. The FDM seeded data range for OFSA_LEAF_DESC is WHERE LEAF_NUM_ID=0 and

LEAF_NODE<10000. If more records are found in this range than the seeded count for FDM version, the Synchronize Instrument procedure displays the error message and terminates. Delete any user-defined Financial Element leaf values within the FDM seeded data range in order to resolve this problem.

Exception 3: Description table does not exist

The exception message reads:

```
WARNING: 'Description Table Name' code table could not be synchronized
due to :ORA-00942: table or view does not exist. These tables must be
synchronized manually. Failure to do so may result in inaccurate
reports.
```

This error occurs while inserting into the description table when user defined values are found in the Code column in dimension member and description table does not exist.

Stage Synchronization

The Stage Synchronization procedure identifies the new Code values from given stage table which will be treated as source and inserts default description entries for them into the appropriate Dimension and CD/MLS tables. The procedure performs both of these synchronizations simultaneously.

OFSAAI requires that all Dimension and Code values have a corresponding description. This is required for any OFSAA reporting operation to return the correct results.

The following topics are covered in this section:

- Tables Requiring Synchronization
- Dimension Member Synchronization
- Code Synchronization
- Executing the Synchronize Stage Procedure
- Exception Messages

Tables Requiring Synchronization

Dimension member Code value synchronization is required only for Stage tables. Required data will be seeded in tables, such as rev_dimensions_b, rev_dimensions_stage_map, and rev_description_tables during installation and before execution of this procedure.

In case, new stage instrument table is being added then insert new set of data (or update the existing data in case of upgrade) in rev_description_tables for non key dimension table and in rev_dimensions_stage_map for key dimension tables.

Dimension Member Synchronization

The SYNCHRONIZE_STAGE procedure synchronizes the dimension member tables and Stage tables using default values for member descriptions and other information columns.

This procedure performs the following functions:

- Checks the specified table (Stage Table) for new dimension members in each of that table's key dimension columns and adds the new dimension value as leaf members to the respective dimension member tables.
- Adds the new dimension member to the corresponding attribute tables with default values for mandatory attributes.
- When new dimension members are added to the dimension tables these members include 'No Description_<ID column Value>' in the DESCRIPTION column and contain default values for mandatory attributes.

Code Synchronization

The SYNCHRONIZE_STAGE procedure identifies code values in stage tables for which a corresponding description does not exist in key dimension or CD/MLS tables and inserts a default description into the appropriate Code Description object. This applies only to CODE columns categorized as **User-Editable** or **User-Defined** (refer table classification).

CODE columns for which OFSAA reserves all of the values are not updated by this procedure. This procedure displays a warning message for any unidentified values in the CODE columns where OFSAA reserves the entire range.

If the resulting objects is a User-Editable or User-Defined Code Description object (checks from REV_TABLE_CLASS_ASSIGNMENT table), then the procedure inserts a default description for any code values for which a description record does not already exist.

If the resulting object is an OFSAA Reserved Code Description object, then the procedure outputs a warning message indicating how many invalid code values exist in the specified Stage table in the message log file **FSI_MESSAGE_LOG**.

For example, if you are synchronizing the STG_INVESTMENTS table, the procedure queries all of the CODE columns on this table. An example of a Reserved CODE column is ACCRUAL_BASIS_CD. If the procedure finds any code values in this column that are not present in the corresponding Code Description object (FSI_ACCRUAL_BASIS_CD), it gives output as an error message indicating the number of invalid values present.

OFSAA Reserved Code Description objects are identified by the following SQL statement:

```
SELECT distinct member_base_table_name, description_table_name
           FROM rev_description_tables,REV_DIMENSIONS_B
   WHERE
rev_description_tables.member_base_table_name=REV_DIMENSIONS_B.Member_b_
Table_Name
       and
rev_description_tables.description_join_column_name=REV_DIMENSIONS_B.MEM
BER_DISPLAY_CODE_COL
   AND stg_table_name = 'STG_INVESTMENTS'
   AND rev_dimensions_b.member_code_column is not null
   AND description_table_name IN
           (SELECT table_name
            FROM rev_table_class_assignment
            WHERE table_classification_cd = 197)
   AND STG_CD_COLUMN_NAME IN
           (SELECT column_name
            FROM user_tab_columns
            WHERE table_name = 'STG_INVESTMENTS');
```

An example of a User-Editable CODE column is BRANCH_CD. If the procedure finds any code values in BRANCH_CD in the FSI_BRANCH_CD table that do not have a description in FSI_BRANCH_MLS, it creates a default description 'No Description_<CD Column Value>' for each value. It is now, up to the users to update these descriptions as appropriate.

User-Editable Code Description objects are identified by the following SQL statement:

```
SELECT distinct member_base_table_name, description_table_name
       FROM rev_description_tables,REV_DIMENSIONS_B,user_tables
WHERE
rev_description_tables.member_base_table_name=REV_DIMENSIONS_B.Member_b_
Table_Name
 and
rev_description_tables.description_join_column_name=REV_DIMENSIONS_B.MEM
BER_DISPLAY_CODE_COL
 and
rev_description_tables.member_base_table_name=user_tables.Table_Name
AND stg_table_name = 'STG_INVESTMENTS'
AND rev_dimensions_b.member_code_column is not null
AND description_table_name NOT IN
            (SELECT table_name
             FROM rev_table_class_assignment
             WHERE table_classification_cd = 197);
```

Executing the Synchronize Stage Procedure

You can execute the SYNCHRONIZE_STAGE procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework. To run the procedure from SQL*Plus, logon to SQL*Plus as the Schema Owner.

The procedure requires 2 parameters: The table name to be synchronized and As of Date. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples.

The syntax for calling the procedure is:

```
Declare
output number;
Begin
fsi_sync_stage('Batch_Id','TABLE_NAME', output)
End;
where table_name is:
• The name of an Stage table
An example of running the stored procedure from SQL*Plus for the
STG_INVESTMENTS table follows:
SQL>var output number;
SQL>fsi_sync_stage('INFODOM_20101231','STG_INVESTMENTS',:output);
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows.

To call the procedure as often as required to synchronize all of your stage tables, the appropriate table name and AS_OF_DATE is enclosed in single quotes.

```
SQL> declare
output number;
begin
fsi_sync_stage('INFODOM_20101231','STG_INVESTMENTS', output);
end;
/
```

To execute the procedure from OFSAAI ICC framework, create a new **Batch** with the task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select the appropriate datastore type from the list.
- Datastore Name:- Select the appropriate datastore name from the list.
- IP address:- Select the IP address from the list.
- Rule Name:- synchronize_stage
- Parameter List:- Stage Table Name

Exception Messages

The SYNCHRONIZE_STAGE procedure may cause some exceptions to appear.

The text messages and explanation for each of these exceptions are as follows:

> **Note:** If you call the procedure from a PL/SQL block, you may want to handle them so that your program can proceed

Exception 1: Description table does not exist

The exception message reads:

WARNING: '**Description Table Name**' code table could not be synchronized due to :ORA-00942: table or view does not exist. These tables must be synchronized manually. Failure to do so may result in inaccurate reports. This error occurs while inserting into

the description table when user defined values are found in the Code column in dimension member and description table does not exist.

Exception 2: Values have been found for Reserved Code Description objects

The exception message reads:

WARNING: Following <count of values> values have been found in <stage table name>.<stage column name> <New found values> for Non Editable Dimension <CD table name>.

Ledger Load Undo

Data loaded into Ledger_Stat table can be undone using the UNDO engine. The following topics are included in this section:

- Parameters for the Undo engine
- Undo mechanism
- Executing Undo engine

Parameters

The following are the parameters to the UNDO engine:

- Batch Run ID (Typical format is INFODOM_BATCHNAME_MISDATE_EXECUTIONSEQUENCE)
- IdentityCode-AsOfDate
- Mode Of Execution

Mode of execution for undoing the ledger load is 'L'. Identity Code and As Of Date are passed in the second parameters with a Hyphen (-) in between.

OFSAAI Batch execution framework is used to invoke the Undo engine.

Undo Mechanism

- Undo Engine will set the STATUS_FLAG column in FSI_DATA_IDENTITY table to 'U' to indicate the start of operation.
- The engine code reads all the records from FSI_DATA_IDENTITY table. For each record that is read, it checks whether

  SOURCE_TYPE = 0

  TABLE_NAME = 'ledger_stat'

IDENTITY_CODE = <as entered by user>, and

AS_OF_DATE = <as entered by user>

After reading all the records from FSI_DATA_IDENTITY table, if a matching record is not found then an error message is logged in the FSI_MESSAGE_LOG table. However, if a matching record is found, then the Undo engine starts the undo process as detailed below.

- Based on the IDENTITY_CODE and Year specified in the AS_OF_DATE, engine prepares and executes an update query to set the amount for the month specified in the AS_OF_DATE to **zero** and attaches a decode statement to calculate the Year To Date amount values from the Period Start month to Period End month. It also attaches any data filter if present to this query.

- Engine also prepares and executes a delete query on LEDGER_STAT table, to delete all the records for which all the month values are 0 and IDENTITY_CODE equals to the value input by user. All entries relevant for the IDENTITY_CODE are also deleted from FSI_DATA_IDENTITY table.

- If the undo fails for any reason, status would be set as 'C'. If Undo is completed successfully, the entry will be removed from FSI_DATA_IDENTITY table.

Executing Undo Engine

To execute the engine from OFSAAI ICC framework, create a new Batch with the Task as RUN EXECUTABLE and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list

- Datastore Name:- Select appropriate name from the list

- IP address:- Select the IP address from the list

- Parameter List:- ./LEDGER_LOAD_UNDO.sh, <Identity Code>-<As_Of_Date>,'L'

To execute the engine from command line, the following is the syntax:

./LEDGER_LOAD_UNDO.sh<parameters>

Parameters: <Batch_Run_Id> <IdentityCode>-<As_of_date> 'L'

Note: AS_OF_DATE should be passed in mm/dd/yyyy format.

Exception Messages

The ledger undo program throws both user defined exceptions and Oracle database related exceptions. These exception messages could be seen in FSI_MESSAGES_LOG

table with the help of the Batch_Run_Id which was used during execution. The exception list includes all possible validations on the parameters that were passed and database related exceptions.

4

Data Loaders

This chapter details the steps involved in executing various data loaders that are available within OFSAA. Data loaders move data from staging layer to processing layer.

This chapter covers the following topics:

- Dimension Loaders
- Simple Dimension Loader
- Historical Rates Data Loader
- Forecast Rate Data Loader
- Prepayment Rate Data Loader
- Stage Instrument Table Loader
- Transaction Summary Table Loader
- Ledger Data Loader
- Cash Flow Loader
- Pricing Management Transfer Rate Population Procedure
- ALMBI Transformation
- Hierarchy Transformation
- Dim Dates Population
- Fact Ledger Stat Transformation
- Financial Element Dimension Population
- Payment Pattern Loader

Dimension Loaders

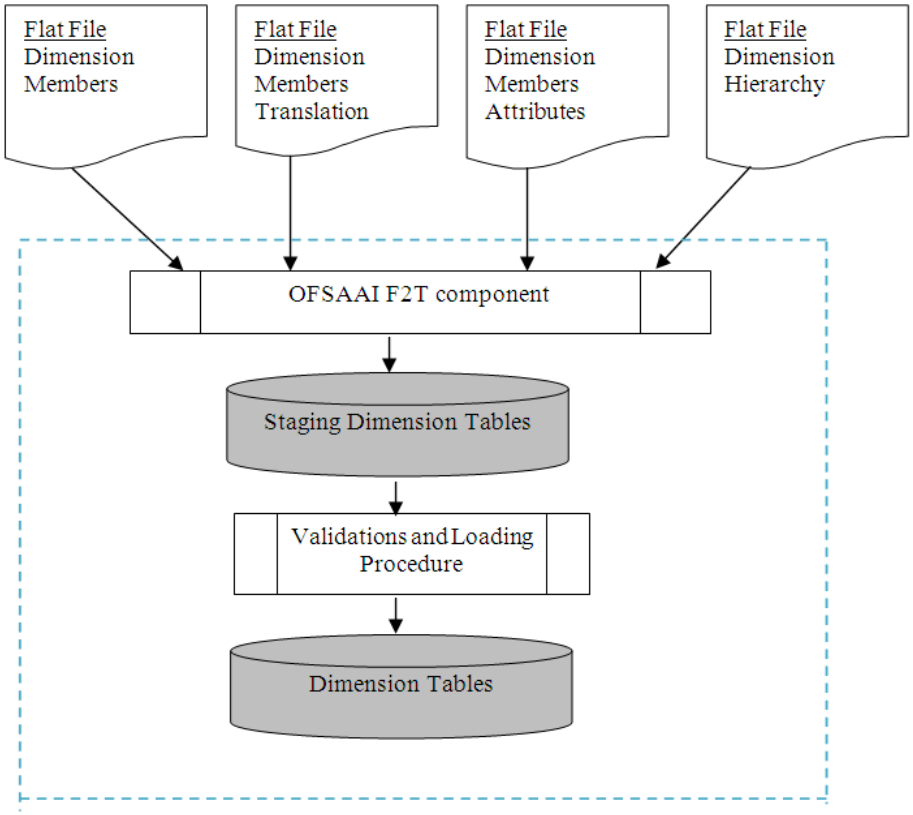
The Dimension Loader procedure populates dimension members, attributes and

hierarchies from Staging dimension tables into dimension tables registered within OFSAAI AMHM framework. Users can view the members and hierarchies loaded by the dimension loader through AMHM screens.

The following topics are covered in this section:

- Overview of Dimension Loaders
- Dimension tables that are part of Staging
- Setting up Loading
- Dimension Loader Procedure
- Executing the Dimension Loader
- Executing the Reverse Population Procedure
- Exception Messages
- Executing the Dimension Load Procedure using the Master Table approach
- Updating DIM_<DIMENSION>_B <Dimension>_Code column with values from DIM_<DIMENSION>_ATTR table
- Executing the Truncate Stage Tables Procedure

Dimension Loader Overview



The dimension loader is used to:

- Load dimension members and their attributes from the staging area into Dimension tables that are registered with the OFSAAI AMHM framework.
- Create hierarchies in AMHM.
- Load hierarchical relationships between members within hierarchies from the staging area into AMHM.

Some of the features of the dimension loader are:

- Multiple hierarchies can be loaded from staging tables.
- Validations of members and hierarchies are similar to that of being performed within AMHM screens.
- Members can be loaded incrementally or fully synchronized with the staging tables.

Enhancements to Support Alphanumeric Code in Dimensions

Note: Dimension Loaders and UIs support capturing an alphanumeric code in addition to the numeric code.

The following Data Model components are required to support dimension member code storage; changes in {6.0/7.3.0/7.3.1} are as follows:

- Release 7.3.1: Dimension Configuration via manual updates to REV_DIMENSIONS_B columns: MEMBER_DATA_TYPE_CODE and MEMBER_CODE_COLUMN. (Also See: OFSAAI Installation & Configuration Guide 7.3 and AI Administration Guide)

- Release 6.0 (7.3) Stage Dimension Interface Table alphanumeric member code column (v_< DIM >_code).

- Release 6.0 (7.3) Stage Dimension Loader Program can directly load alphanumeric member codes.

For further details on display of member codes in the user interfaces, see OFSAAI User Guide 7.3.

Tables that are Part Of Staging

Dimension data is stored in the following set of tables:

- STG _<DIMENSION>_B_INTF - Stores leaf and node member codes within the dimension.

- STG_<DIMENSION>_ TL_INTF - Stores names of leaf and node and their translations.

- STG_<DIMENSION>_ ATTR_INTF - Stores attribute values for the attributes of the dimension.

- STG_<DIMENSION>_ HIER_INTF - Stores parent-child relationship of members and nodes that are part of hierarchies.

- STG_HIERARCHIES_INTF - Stores master information related to hierarchies.

Data present in the above set of staging dimension tables are loaded into the below set of dimension tables.

- DIM_<DIMENSION>_ B - Stores leaf and node member codes within the dimension.

- DIM_<DIMENSION>_TL - Stores names of leaf and node and their translations.

- DIM_<DIMENSION>_ATTR - Stores attribute values for the attributes of the dimension.
- DIM_<DIMENSION>_HIER - Stores parent-child relationship of members and nodes that are part of hierarchies.
- REV_HIERARCHIES - Stores hierarchy related information.
- REV_HIERARCHY_LEVELS - Stores levels of the hierarchy.

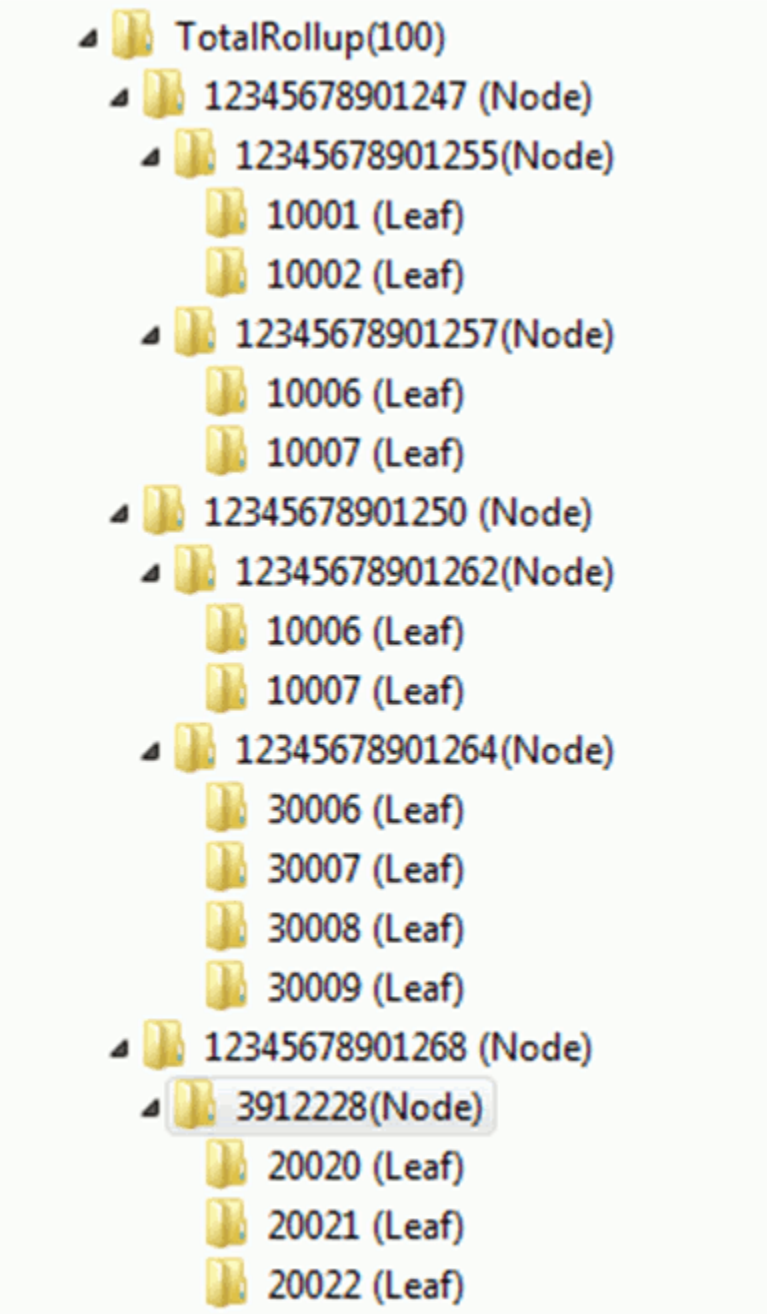
Staging tables are present for all key dimensions that are configured within the OFSAAI framework. For any custom key dimension that is added by the Client, respective staging dimension tables like STG_<DIMENSION>_B_INTF, STG_< DIMENSION>_TL_INTF, STG_<DIMENSION>_ATTR_INTF, and STG_<DIMENSION>_HIER_INTF have to be created in the ERwin model.

Populating STG_<DIMENSION>_HIER_INTF Table

The STG_<DIMENSION>_HIER_INTF table is designed to hold hierarchy structure. The hierarchy structure is maintained by storing the parent child relationship in the table. In the following hierarchy there are 4 levels. The first level node is 100, which is the Total Rollup. The Total Rollup node will have the N_PARENT_DISPLAY_CODE and N_CHILD_DISPLAY_CODE as the same.

Column Name	Column Description
V_HIERARCHY_OBJECT_NAME	Stores the name of the hierarchy
N_PARENT_DISPLAY_CODE	Stores the parent Display Code
N_CHILD_DISPLAY_CODE	Stores the child Display Code
N_DISPLAY_ORDER_NUM	Determines the order in which the structure (nodes, leaves) of the hierarchy should be displayed. This is used by the UI while displaying the hierarchy. There is no validation to check if the values in the column are in proper sequence.
V_CREATED_BY	Stores the created by user. Hard coded as -1
V_LAST_MODIFIED_BY	Stores the last modified by user. Hard coded as -1

Hierarchy Structure



Sample Data

V_HIERARCHY_OBJECT_NAME	N_PARENT_DISPLAY_CODE	N_CHILD_DISPLAY_CODE	N_DISPLAY_ORDER_NUM	V_CREATED_BY	V_LAST_MODIFIED_BY
INCOME STMT	100	100	1	-1	-1
INCOME STMT	100	12345678901247	2	-1	-1
INCOME STMT	12345678901247	12345678901255	1	-1	-1
INCOME STMT	12345678901255	10001	1	-1	-1
INCOME STMT	12345678901255	10002	2	-1	-1
INCOME STMT	12345678901247	12345678901257	2	-1	-1
INCOME STMT	12345678901257	10006	1	-1	-1
INCOME STMT	12345678901257	10007	2	-1	-1
INCOME STMT	100	12345678901250	3	-1	-1
INCOME STMT	12345678901250	12345678901262	2	-1	-1
INCOME STMT	12345678901262	30005	1	-1	-1
INCOME STMT	12345678901250	12345678901264	1	-1	-1
INCOME STMT	12345678901264	30006	1	-1	-1
INCOME STMT	12345678901264	30007	2	-1	-1
INCOME STMT	12345678901264	30008	3	-1	-1
INCOME STMT	12345678901264	30009	4	-1	-1
INCOME STMT	100	12345678901268	4	-1	-1
INCOME STMT	12345678901268	3912228	1	-1	-1
INCOME STMT	3912228	20020	1	-1	-1

V_HIERARCHY_OBJECT_NAME	N_PARENT_DISPLAY_CODE	N_CHILD_DISPLAY_CODE	N_DISPLAY_ORDER_NUM	V_CREATED_BY	V_LAST_MODIFIED_BY
INCOME STMT	3912228	20021	2	-1	-1
INCOME STMT	3912228	20022	3	-1	-1

Column REV_DIMENSIONS_B.MEMBER_CODE_COLUMN

In release 7.3.1: With the introduction of alphanumeric support, REV_DIMENSIONS_B.MEMBER_CODE_COLUMN column becomes important for successful execution of the dimension loader program and subsequent T2Ts. The value in this column should be a valid code column from the relevant DIM_< DIMENSION>_B (key dimension) or FSI_< DIM>_CD (simple dimension) table. The Leaf_registration procedure currently does not populate this column. The values should be updated manually with the correct DIM_< DIM>_B.< DIM>_CODE or FSI_< DIM>_CD.< DIM>_CD column. Setting this will ensure that the values in this column are displayed for both numeric and alphanumeric dimensions as "Alphanumeric Code" in the UI. Configuration of an alphanumeric dimension also requires manual update of the REV_DIMENSIONS_B. MEMBER_DATA_TYPE_CODE column.

For more information, see OFSAAI Installation & Configuration Guide.

Dimension Load Procedure

This procedure performs the following functions:

- Gets the list of source and target dimension tables. The dimension tables for a given dimension are stored in REV_DIMENSIONS_B table. The stage tables for a given dimension are stored in FSI_DIM_LOADER_SETUP_DETAILS.

- The parameter Synchronize Flag can be used to completely synchronize data between the stage and the dimension tables. If the flag = 'Y' members from the dimension table which are not present in the staging table will be deleted. If the flag is 'N' the program merges the data between the staging and dimension table.

- The Loader program validates the members/attributes before loading them.
    - The program validates the number of records in the base members table - STG_<DIMENSION>_B_INTF and translation members table - STG_<DIMENSION>_TL_INTF. The program exits if the number of records does not match.

    - In case values for mandatory attributes are not provided in the staging tables, the loader program populates the default value (as specified in the attribute

maintenance screens within AMHM of OFSAAI) in the dimension table.

- The program validates for data types of attribute value. For example an attribute that is configured as 'NUMERIC' cannot have non-numeric values.
- Dimension Loader validates the attribute against their corresponding dimension table. If any of the attributes is not present, then an error message will be logged in FSI_MESSAGE_LOG table.
- Dimension Loader will check the number of records in Dim_<Dim_Name>_B and Dim_<Dim_Name>_TL for the language. In case any mismatch is found, then an error will be logged and loading will be aborted.

- If all the member level validations are successful the loader program inserts the data from the staging tables to the dimension tables.

  Note: In release 6.0 (7.3?) The stage dimension loader program is modified to move alphanumeric code values from STG_< DIMENSION >_B_INTF.V_< DIM >_CODE to DIM_< DIM >_B.< DIM >_CODE column. Previously, the DIM_< DIM>_B.< DIM >_CODE column was populated using the fn_updateDimensionCode procedure from the code attributes. With this enhancement users can directly load alphanumeric values.

  The fn_updateDimensionCode procedure is still available for users who do not want make any changes to their ETL procedures for populating the dimension staging tables (e.g. STG_< DIMENSION >_B_INTF,, STG_< DIMENSION>_ATTR_INTF).

- After this, the loader program loads hierarchy data from staging into hierarchy tables.
- In case of hierarchy data the loader program validates if the members used in the hierarchy are present in the STG_<DIMENSION>_B_INTF table.
- The program validates if the hierarchy contains multiple root nodes and logs error messages accordingly, as multiple root nodes are not supported.
- Dimension Loader will check special characters in Hierarchy. Hierarchy name with special characters will not be loaded.

  Following are the list of special characters which are not allowed in Hierarchy Name:

  ^&\'"

After execution of the dimension loader, the user must execute the reverse population procedure to populate OFSA legacy dimension and hierarchy tables.

Setting up Dimension Loader

FSI_DIM_LOADER_SETUP_DETAILS table should have record for each dimension that has to be loaded using the dimension loader. The table contains seeded entries for key dimensions that are seeded with the application.

The following are sample entries in the setup table:

Column Name	Description	Sample Value
n_dimension_id	This stores the Dimension ID	1
v_intf_b_table_name	Stores the name of the Staging Base table	Stg_org_unit_b_intf
v_intf_member_column	Stores the name of the Staging Member Column Name	V_org_unit_id
v_intf_tl_table_name	Stores the name of the Staging Translation table	Stg_org_unit_tl_intf
v_intf_attr_table_name	Stores the name of the Staging Member Attribute table	Stg_org_unit_attr_intf
v_intf_hier_table_name	Stores the name of the Staging Hierarchy table	Stg_org_unit_hier_intf
d_start_time	Start time of loader - updated by the loader program.	
d_end_time	End time of loader - updated by the loader program.	
v_comments	Stores Comments.	Dimension loader for organization unit.
v_status	Status updated by the Loader program.	
v_intf_member_name_col	Stores the name of the Member	V_org_unit_name

Column Name	Description	Sample Value
v_gen_skey_flag	Flag to indicate if surrogate key needs to be generated for alphanumeric codes in the staging. Applicable only for loading dimension data from master tables. Not applicable for loading dimension data from interface tables.	
v_stg_member_column	Name of the column that holds member code in the staging table. Applicable for loading dimension data from both master tables and interface tables." (sample value "v_org_unit_code ") – this appears to be the alphanumeric code	v_org_unit_code
v_stg_member_name_col	Name of the column that holds member name in the staging table. Applicable only for loading dimension data from master tables. Not applicable for loading dimension data from interface tables.	
v_stg_member_desc_col	Name of the column that holds description in the staging table. Applicable only for loading dimension data from interface tables.	

Executing the Dimension Load Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires 4 parameters – Batch Run Identifier, As of Date, Dimension Identifier, Synchronize flag (Optional). The syntax for calling the procedure is:

```
function fn_drmDataLoader(batch_run_id varchar2,
                          as_of_date varchar2,
                          pDimensionId varchar2,
                          pSynchFlag char default 'Y')
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- AS_OF_DATE in the format YYYYMMDD.
- pDIMENSIONID dimension id.

- pSynchFlag this parameter is used to identify if a complete synchronization of data between staging and dimension table is required. The default value is 'Y'.

  Note: With Synch flag N, data is moved from Stage to Dimension tables. Here, an appending process happens. You can provide a combination of new Dimension records plus the data that has undergone change. New records are inserted and the changed data is updated into the Dimension table.

  With Synch flag Y, the Stage table data will completely replace the Dimension table data.

  There are a couple of checks in place to ensure that stage_dimension_loader is equipped with similar validations that the UI provides.

  The Data Loader does a Dependencies Check before a member is deleted. The validation checks, if there are members used in the Hierarchy that are not present in the DIM_< DIM >_B table. This is similar to the process of trying to delete a member from the UI, which is being used in the Hierarchy definition. You are expected to remove or delete such Hierarchies from the UI before deleting a member.

For Example:

```
Declare
   num number;
Begin
   num := fn_drmDataLoader ('INFODOM_20100405','20100405' ,1,'Y' );
End;
```

To execute the procedure from the OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- *fn_drmDataLoader*
- Parameter List:- Dimension ID, Synchronize Flag

The fn_drmdataloader function calls STG_DIMENSION_LOADER package which loads data from the stg_<dimension>_hier_intf to the dim_<dimension>_hier table.

Exception Messages

The text and explanation for each of these exceptions follows. If you call the procedure

from a PL/SQL block you may want to handle these exceptions appropriately so that your program can proceed without interruption.

Exception 1: Error. errMandatoryAttributes

This exception occurs when the stage Loader program cannot find any data default value for mandatory attributes.

Exception 2: Error. errAttributeValidation

This exception occurs when there is a data type mis-match between the attribute value and configured data-type for the attribute.

Exception 3: Error. errAttributeMemberMissing

If there is a mismatch in the count between the member's base and translation table.

Executing the Dimension Load Procedure using Master Table approach

FSI_DIM_LOADER_SETUP_DETAILS table should have a record for each dimension that has to be loaded. The table contains entries for key dimensions that are seeded with the application.

The following columns must be populated for user-defined Dimensions.

v_stg_member_column

v_stg_member_name_col

v_stg_member_desc_col

Additionally, the FSI_DIM_ATTRIBUTE_MAP table should be configured with column attribute mapping data. This table maps the columns from a given master table to attributes.

N_DIMENSION_ID	This stores the Dimension ID
V_STG_TABLE_NAME	This holds the source Stage Master table
V_STG_COLUMN_NAME	This holds the column from the master table
V_ATTRIBUTE_NAME	This holds the name of the attribute the column maps to

V_UPDATE_B_CODE_FLAG	This column indicates if the attribute value can be used to update the code column in the DIM_<Dimension>_B table. **Note:** fn_STGDimDataLoader does not use FSI_DIM_ATTRIBUTE_MAP.V_UPDATE_B_CODE_FLAG

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the ICC Batch screen within OFSAAI framework. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires 5 parameters: – Batch Run Identifier , As of Date, Dimension Identifier , MIS-Date Required Flag, Synchronize flag (Optional). The syntax for calling the procedure is:

```
function fn_STGDimDataLoader(batch_run_id varchar2,
                             as_of_date varchar2,
                             pDimensionId varchar2,
                             pMisDateReqFlag char default 'Y',
                             pSynchFlag char default 'N')
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- AS_OF_DATE in the format YYYYMMDD.
- pDIMENSIONID dimension id.
- pMisDateReqFlag is used to identify if AS-OF_DATE should be used in the where clause to filter the data.
- pSynchFlag is used to identify if a complete synchronization of data between staging and fusion table is required. The default value is 'Y'.

For Example

```
Declare
   num number;
Begin
  num := fn_STGDimDataLoader ('INFODOM_20100405','20100405' ,1,'Y','Y'
);
End;
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list

- IP address:- Select the IP address from the list
- Rule Name:- *fn_STGDimDataLoader*
- Parameter List:- Dimension ID, Mis Date Required Flag , Synchronize Flag

Clients could face a problem while loading customer dimension into AMHM using the Master table approach.

Configuring the setup table for CUSTOMER dimension is pretty confusing while dealing with attributes like FIRST_NAME , MIDDLE_NAME and LAST_NAME.

Most clients would like to see FIRST_NAME , MIDDLE_NAME and LAST_NAME forming the name of the member within the customer dimension.

Currently the STG_DIMENSION_LOADER disallows concatenation of columns.

Moreover the concatenation might not ensure unique values.

As a solution to this problem we can work on the following options:

Approach 1

1. Create a view on STG_CUSTOMER_MASTER table with FIRST_NAME, MIDDLE_NAME and LAST_NAME concatenated and identify this column as NAME.
2. Configure the name column from the view in FSI_DIM_LOADER_SETUP_DETAILS
3. Increase the size of DIM_CUSTOMER_TL.name column.
4. Disable the unique index on DIM_CUSTOMER_TL.NAME or append Customer_code to the NAME column.
5. The NAME column will be populated into the DIM_CUSTOMER_TL.NAME column.

Approach 2

Populate customer_code into the DIM_CUSTOMER_TL.NAME column.

Updating DIM_<DIMENSION>_B <Dimension>_Code column with values from DIM_<DIMENSION>_ATTR table

The stage dimension loader procedure does not insert or update the <Dimension>_code column in the Dim_<Dimension>_B table. This is an alternate method for updating the < Dimension>_Code column in the Dim_< Dimension>_B table, retained to accommodate implementations prior to the enhancement where we enable loading the code directly to the dimension table instead of from the attribute table. It is not recommended for new

installations. This section explains how the <Dimension>_code can be updated.

Steps to be followed

1. A new attribute should be created in the REV_DIM_ATTRIBUTES_B / TL table.

   Note: You should use the existing "CODE" attribute for the seeded dimensions.

   Example
   PRODUCT CODE, COMMON COA CODE and so on.

2. The fsi_dim_attribute_map table should be populated with values.

   The following columns must be populated:

   N_DIMENSION_ID (Dimension id)

   V_ATTRIBUTE_NAME (The attribute name)

   V_UPDATE_B_CODE_FLAG (This flag should be 'Y'). Any given dimension can have only one attribute with V_UPDATE_B_CODE_FLAG as 'Y'. This should only be specified for the CODE attribute for that dimension.

   Example:

N_DIMENSION_ID	4
V_ATTRIBUTE_NAME	'PRODUCT_CODE'
V_UPDATE_B_CODE_FLAG	'Y'
V_STG_TABLE_NAME	'stg_product_master'
V_STG_COLUMN_NAME	'v_prod_code'

   Note: The values in V_STG_TABLE_NAME and V_STG_COLUMN_NAME are not used by the fn_updateDimensionCode procedure, however these fields are set to NOT NULL and should be populated.

3. Load STG_<DIMENSION>_ATTR_INTF table with data for the new ATTRIBUTE created.

   Note: The attribute values must first be loaded using the stage

dimension loader procedure, fn_drmDataLoader, before running this procedure. This procedure will pull values from the DIM_<DIMENSION>_ATTR table. If these rows do not exist for these members prior to running this procedure, the DIM_<DIMENSION>_B.<DIMENSION>_CODE field will not be updated.

4. Execute the fn_updateDimensionCode function. The function updates the code column with values from the DIM_<DIMENSION>_ATTR table.

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Atomic Schema Owner. The function requires 3 parameters – Batch Run Identifier , As of Date, Dimension Identifier. The syntax for calling the procedure is:

```
function fn_updateDimensionCode (batch_run_id varchar2,
                                 as_of_date varchar2,
                                 pDimensionId varchar2)
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- AS_OF_DATE in the format YYYYMMDD.
- pDIMENSIONID dimension id

For Example

```
Declare
      num number;
Begin
     num := fn_updateDimensionCode ('INFODOM_20100405','20100405',1 );
End;
```

You need to populate a row in FSI_DIM_LOADER_SETUP_DETAILS.

For example, for FINANCIAL ELEM CODE, to insert a row into FSI_DIM_LOADER_SETUP_DETAILS, following is the syntax:

INSERT INTO FSI_DIM_LOADER_SETUP_DETAILS (N_DIMENSION_ID) VALUES ('0'); COMMIT;

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list

- Rule Name:- *Update_Dimension_Code*
- Parameter List:- Dimension ID

Truncate Stage Tables Procedure

This procedure performs the following functions:

- The procedure queries the FSI_DIM_LOADER_SETUP_DETAILS table to get the names of the staging table used by the Dimension Loader program.
- The MIS Date option only works to the "Master Table approach" (fn_STGDimDataLoader) dimension loader. It is not applicable to dimension data loaded using the standard Dimension Load Procedure (fn_drmDataLoader).

Executing the Truncate Stage Tables Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires 4 parameters – Batch Run Identifier, As of Date, Dimension Identifier, Mis Date Required Flag. The syntax for calling the procedure is:

```
function fn_truncateStageTable(batch_run_id varchar2,
                                as_of_date varchar2,
                                pDimensionId varchar2,
                                pMisDateReqFlag char default 'Y')
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- AS_OF_DATE in the format YYYYMMDD.
- pDIMENSIONID dimension id.
- pMisDateReqFlag is used to identify the data needs to be deleted for a given MIS Date. The default value is 'Y'.

For Example

```
Declare
       num number;
Begin
   num := fn_truncateStageTable ('INFODOM_20100405','20100405' ,1,'Y' );
End;
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type:- Select appropriate datastore from list

- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- *fn_truncateStageTable*
- Parameter List:- Dimension ID, MIS-Date required Flag

Simple Dimension Loader

Currently the dimension loader program works only for key dimensions.

Simple Dimension Loader provides the ability to load data from stage tables to Simple dimension tables.

For example, the user can load data into FSI_ACCOUNT_OFFICER_CD and FSI_ACCOUNT_OFFICER_MLS using the Simple Dimension Loader program.

Simple dimension of type 'writable and editable' can use this loading approach. This can be identified by querying rev_dimensions_b.write_flag = 'Y', rev_dimensions_b.dimension_editable_flag ='Y' and rev_dimensions_b.simple_dimension_flag = 'Y'.

The following topics are covered in this section:

- Creating Simple Dimension Stage Table
- Configuration of Setup Tables
- Executing the Simple Dimension Load Procedure
- Exception Messages

Creating Simple Dimension Stage Table

You can create stage tables for the required simple dimensions by using the following template:

STG_<DIM>_MASTER			
COLUMN_NAME	**DATA TYPE**	**PRIMARY KEY**	**NULLABLE**
v_< DIM>_display_code	Varchar2(10)	Y	N
d_Mis_date	Date	Y	N

STG_<DIM>_MASTER			
COLUMN_NAME	**DATA TYPE**	**PRIMARY KEY**	**NULLABLE**
v_Language	Varchar2(10)	Y	N
v_< DIM>_NAME	Varchar2(40)		N
v_Description	Varchar2(255)		N
v_Created_by	Varchar2(30)		Y
v_Modified_by	Varchar2(30)		Y

Here is a sample structure:

STG_ACCOUNT_OFFICER_MASTER			
COLUMN_NAME	**DATA TYPE**	**PRIMARY KEY**	**NULLABLE**
v_acct_officer_display_code	Varchar2(10)	Y	N
d_Mis_date	Date	Y	N
v_Language	Varchar2(10)	Y	N
v_Name	Varchar2(40)		N
v_Description	Varchar2(255)		N
v_Created_by	Varchar2(30)		Y
v_Modified_by	Varchar2(30)		Y

Here are some examples:

- Example For FSI CD/MLS tables:

```
CREATE TABLE <XXXXX>_FSI_<DIM>_CD -- ACME_FSI_ACCT_STATUS_CD
(<DIM>_CD NUMBER(5) -- ACCT_STATUS_CD
,LEAF_ONLY_FLAG VARCHAR2(1)
,ENABLED_FLAG VARCHAR2(1)
,DEFINITION_LANGUAGE VARCHAR2(10)
,CREATED_BY VARCHAR2(30)
,CREATION_DATE DATE
,LAST_MODIFIED_BY VARCHAR2(30)
,LAST_MODIFIED_DATE DATE
<dim>_display_CD VARCHAR2(10)
);
```

- Example for FSI_<DIM>_MLS table:

```
CREATE TABLE <XXXXX>_FSI_<DIM>_MLS -- ACME_FSI_ACCT_STATUS_CD
(<DIM>_CD NUMBER(5) -- ACCT_STATUS_CD
,LANGUAGE VARCHAR2(10)
,<DIM> VARCHAR2(40) -- ACCT_STATUS
,DESCRIPTION VARCHAR2(255)
,CREATED_BY VARCHAR2(30)
,CREATION_DATE DATE
,LAST_MODIFIED_BY VARCHAR2(30)
,LAST_MODIFIED_DATE DATE
);
```

Note: FSI_<DIM>_CD and FSI_<DIM>_MLS should follow the same standards as mentioned above, else Loader will not work as expected.

Configuration of Setup Tables

REV_DIMENSIONS_BTable

The REV_DIMENSIONS_B table holds the following target table information:

The target FSI_<DIM>_CD/MLS table can be retrieved from REV_DIMENSIONS_B table as follows:

dimension_id ---> Holds the id of the simple dimension that needs to be loaded.

member_b_table_name ---> Holds the name of the FSI_<DIM>_CD target table. For example, FSI_ACCOUNT_OFFICER_CD

member_tl_table_name ---> Holds the name of the FSI_<DIM>_MLS table name. For example, FSI_ACCOUNT_OFFICER_MLS

member_col ---> Holds the Column Name for which Surrogate needs to be generated. For example, ACCOUNT_OFFICER_CD

member_code_column ---> Holds the Name of the joining column name from FSI_<DIM>_CD Display code column. For example, ACCOUNT_OFFICER_DISPLAY_CD

key_dimension_flag ---> N

dimension_editable_flag ---> Y

write_flag ---> Y

simple_dimension_flag ---> Y

Setup Table Configuration Mapping

The FSI_DIM_LOADER_SETUP_DETAILS stores the STG_<DIM>_MASTER table details as follows:

FSI_DIM_LOADER_SETUP_DETAILS	STG_<DIM>_MASTER
N_DIMENSION_ID	<dimension_id> For example, 617
V_INTF_B_TABLE_NAME	Stage table name For example, STG_ACCOUNT_OFFICER_MASTER
V_GEN_SKEY_FLAG	Default will be 'Y', it generates Surrogate Key. When 'N' then stage display code column will be used as a surrogate key. For example, FSI_ACCOUNT_OFFICER_CD.ACCOUNT_OFFICER_DISPLAY_CD should be numeric.
V_STG_MEMBER_COLUMN	Stores the stage display code column. For example, STG_ACCOUNT_OFFICER_MASTER.v_acct_officer_display_code
V_STG_MEMBER_NAME_COL	Stores the stage column name. For example, STG_ACCOUNT_OFFICER_MASTER.v_Name
V_STG_MEMBER_DESC_COL	Stores the stage description column name. For example, STG_ACCOUNT_OFFICER_MASTER.v_description

Executing the Simple Dimension Load Procedure

There are two ways to execute the simple dimension load procedure:

- **Running Procedure Using SQL*Plus**

  To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

```
function fn_simpledimloader(batch_run_id VARCHAR2, as_of_date
VARCHAR2, pdimensionid VARCHAR2,
pMisDateReqFlag char default 'Y', psynchflag CHAR DEFAULT 'N')

SQLPLUS > declare
result number;
begin
result := fn_simpledimloader
('SimpleDIIM_BATCH1','20121212','730','N','Y');
end;
/
```

- BATCH_RUN_ID is any string to identify the executed batch.
- AS_OF_DATE is in the format YYYYMMDD.
- pDIMENSIONID is the dimension ID.
- pSynchFlag this parameter is used to identify if a complete synchronization of data between staging and dimension table is required. The default value is 'Y'.

  pMisDateReqFlag : Filter will be placed on the input stage table to select only the records which falls on the given as_of_date. Default value is Y. If complete stage table data needs to be considered, then it should be passed 'N'.

  > **Note:** With Synch flag N, data is moved from Stage to Dimension tables. Here, an appending process happens. You can provide a combination of new Dimension records plus the data that has undergone change. New records are inserted and the changed data is updated into the Dimension table. With Synch flag Y, the Stage table data will completely replace the Dimension table data.

- **Simple Dimension Load Procedure Using OFSAAI ICC Framework**

  To execute Simple Dimension Loader from OFSAAI ICC framework, a seeded Batch is provided.

  The batch parameters are:

  - Datastore Type:- Select the appropriate datastore from list
  - Datastore Name:- Select the appropriate name from the list
  - IP address:- Select the IP address from the list
  - Rule Name:- fn_simpledimloader
  - Parameter : 'Pass the dimension id for which DT needs to be executed, psynchflag'

For example, '730,N,Y'

Note: In case of FSI_ACCOUNT_OFFICER_CD query:

SELECT dimension_id FROM rev_dimensions_b where member_b_table_name = 'FSI_ACCOUNT_OFFICER_CD'

pass the dimension_id.

- Psynchflag:- By default it is N, data is moved from Stage to Dimension tables. Here, an appending process happens. You can provide a combination of new Dimension records plus the data that has undergone change. New records are inserted and the changed data is updated into the Dimension table. With Synch flag 'Y', the Stage table data will completely replace the Dimension table data.

Exception Messages

Below are the list of error messages which can be viewed in view log from UI or fsi_messge_log table from back end filtering for the given batch id. On successful completion of each task, messages gets into log table.

In the event of failure, following are the list of errors that may occur during the execution:

Exception 1: When REV_DIMENSIONS_B is not having proper setup details.

Meaning: For Simple Dimension write_flag, simple_dimension_flag, dimension_editable_flag should be Y in rev_dimensions_b for the given Dimension id.

Exception 2: When FSI_DIM_LOADER_SETUP_DETAILS table is not having proper set up details.

Meaning: Setup details are not found for the dimension id.

Exception 3: When Display code Column is non numeric and trying to use as a surrogate key.

Meaning: Display code Column should be numeric as v_gen_skey_flag N

Historical Rates Data Loader

Historical data for currency exchange rates, interest rates and economic indicators can be loaded into the OFSAA historical rates tables through the common staging area. The T2T component within OFSAAI framework is used to move data from the Stage historical rate tables into the relevant OFSAA processing tables. After loading the rates, users can view the historical rate data through the OFSAA Rate Management UI's.

The following topics are covered in this section:

- Tables related to Historical Rates

- Populating Historical Rate Stage tables
- Executing the Historical Rates Data Loaders
- Re-loading historical rates
- Exception Messages

Tables Related to Historical Rates

Historical rates are stored in the following staging area tables:

- STG_EXCHANGE_RATE_HIST – This staging table contains the historical exchange rates for Currencies used in the system.
- STG_IRC_RATE_HIST - This staging table contains the historical interest rates for the Interest Rate codes used in the system.
- STG_IRC_TS_PARAM_HIST – This staging table contains the historical interest rate term structure parameters, used by the Monte Carlo engine.
- STG_ECO_IND_HIST_RATES - This staging table stores the historical values for the Economic Indicators used in the system.

Historical rates in OFSAA Rate Management are stored in the following processing tables:

- FSI_EXCHANGE_RATE_HIST – This table contains the historical exchange rates for the Currencies used in the system.
- FSI_IRC_RATE_HIST – This table contains the historical interest rates for the Interest Rate codes used in the system.
- FSI_IRC_TS_PARAM_HIST – This table stores the historical interest rate term structure parameters, used by the Monte Carlo engine.
- FSI_ECO_IND_HIST_RATES – This table contains the historical values for the Economic Indicators used in the system.

Populating Stage Tables

Data for historical rates commonly comes from external systems. Such data must be converted into the format of the staging area tables. This data can be loaded into the staging area using the F2T component of the OFSAAI framework. Users can view the loaded data by querying the staging tables and various log files associated with the F2T component.

Executing the Historical Rates Data Loader T2T

There are four pre-defined T2T mappings configured and seeded in OFSAA for the purpose of loading historical rates. These can be executed from the ICC framework within OFSAAI.

To execute the Historical Exchange Rates Data Loader, create a new Batch and specify the following parameters:

- Datastore Type:- Select appropriate datastore from the drop down list
- Datastore Name:- Select appropriate name from the list. Generally it is the Infodom name.
- IP address:- Select the IP address from the list
- Rule Name:- T2T_EXCHANGE_RATE_HIST
- Parameter List:- No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

To execute the Historical Interest Rates Data Loader, create a new Batch and specify the following parameters:

- Datastore Type:- Select appropriate datastore from the drop down list
- Datastore Name:- Select appropriate name from the drop down list
- IP address:- Select the IP address from the list
- Rule Name:- T2T_IRC_RATE_HIST
- Parameter List: No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

To execute the Historical Term Structure Parameter Data Loader, create a new Batch and specify the following parameters:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- T2T_IRC_TS_PARAM_HIST
- Parameter List: No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

To execute the Historical Economic Indicator Data Loader, create a new Batch and specify the following parameters:

- Datastore Type:- Select appropriate datastore from the drop down list
- Datastore Name: - Select appropriate name from the drop down list
- IP address:- Select the IP address from the list
- Rule Name:- T2T_ECO_IND_HIST_RATES
- Parameter List: No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

After executing any of the above batch processes, check the T2T component logs and batch messages to confirm the status of the data load.

The T2T component can fail under the following scenario:

- Unique constraint error – Target table may already contain data with the primary keys that the user is trying to load from the staging area.

Re-Load Of Historical Rates

The T2T component can only perform "Insert" operations. In case the user needs to perform updates, previously loaded records should be deleted before loading the current records. Function fn_deleteFusionTables is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function requires entries in the FSI_DELETE_TABLES_SETUP table to be configured. Configure the below table for all columns that need to be part of the join between the Stage table and Equivalent table.

Users can create new or use existing Data Transformations for deleting a Table. The parameters for the Data Transformation are:

- 'Table to be deleted'
- Batch run ID
- As of Date

Column Name	Column Description	Sample Value
STAGE_TABLE_NAME	Stores the source table name for forming the join statement	STG_LOAN_CONTRACTS

STAGE_COLUMN_NAME	Stores the source column name for forming the join statement	V_ACCOUNT_NUMBER
FUSION_TABLE_NAME	Stores the target table name for forming the join statement	FSI_D_LOAN_CONTRACTS
FUSION_COLUMN_NAME	Stores the target column name for forming the join statement	ACCOUNT_NUMBER

Note: Insert rows in FSI_DELETE_TABLES_SETUP for all columns that can be used to join the stage with the equivalent table. In case if the join requires other dimension or code tables, a view can be created joining the source table with the respective code tables and this view can be part of the above setup table.

Forecast Rate Data Loader

The Forecast Rate Data Loader procedure loads forecast rates into the OFSAA ALM Forecast rates processing area tables from staging tables. In ALM, Forecast Rate assumptions are defined within the Forecast Rate Assumptions UI. The Forecast Rates Data Loader supports the Direct Input and Structured Change methods only for exchange rates, interest rates and economic indicators. Data for all other forecast rate methods should be input through the User Interface. After executing the forecast rates data loader, users can view the information in the ALM - Forecast Rates Assumptions UI.

The following topics are covered in this section:

- Tables related to Forecast Rate Data Loader
- Populating Forecast Rate Stage tables
- Forecast Rate Loader Program
- Executing the Forecast Rate Data Loader
- Exception Messages

Forecast Rate Data Loader Tables

Forecast rate assumption data is stored in the following staging area tables:

- STG_FCAST_XRATES – This table holds the forecasted exchange rate data for the current ALM modeling period.
- STG_FCAST_IRCS - This table holds the forecasted interest rate data for the current ALM modeling period.
- STG_FCAST_EI - This table holds the forecasted economic indicator data for the current ALM modeling period.

Rates present in the above staging tables are copied into the following ALM metadata tables.

- FSI_FCAST_IRC_DIRECT_INPUT, FSI_FCAST_IRC_STRCT_CHG_VAL.
- FSI_FCAST_XRATE_DIRECT_INPUT, FSI_FCAST_XRATE_STRCT_CHG.
- FSI_FCAST_EI_DIRECT_INPUT, FSI_FCAST_EI_STRCT_CHG_VAL

Populating Forecast Rate Stage Tables

STG_FCAST_EI

v_forecast_name	The Name of the Forecast Rate assumption rule as defined. The Forecast name indicates the Short Description for the Forecast Rate Sys ID as stored in the FSI_M_OBJECT_DEFINITION_TL table. In case the forecast sys id is provided, then populate this field with -1.
v_scenario_name	This field indicates the Scenario Name for which the Forecast Rate data is applicable.
v_economic_indicator_name	This field indicates the Economic Indicator Name for which the Forecast data is applicable.
n_from_bucket	This field indicates the Start Bucket Number for the given scenario.

fic_mis_date	This field indicates the current period As of Date applicable to the data being loaded.
n_fcast_rates_sys_id	The System Identifier of the forecast rate assumption rule to which this data will be loaded. In case forecast name and folder are provided, then populate this field with -1.
v_folder_name	Name of the folder that holds the Forecast Rate assumption rule definition. In case the forecast sys id is provided, then populate this field with -1.
v_ei_method_cd	The Forecast method of economic indicator values include: Direct Input or Structured change. Use **DI** - For Direct Input or **SC** - For Structured Change
n_economic_indicator_value	This field indicates the value for the Economic Indicator for the given scenario and time bucket.
n_to_bucket	This field indicates the End Bucket Number for the assumption.

STG_FCAST_XRATES

v_forecast_name	The Name of the Forecast Rate assumption rule as defined. The Forecast name indicates the Short Description for the Forecast Rate Sys ID as stored in the FSI_M_OBJECT_DEFINITION_TL table. In case the forecast sys id is provided, then populate this field with -1.

v_scenario_name	This field indicates the Scenario Name for which the Forecast Rate data is applicable.
v_iso_currency_cd	From ISO Currency Code (like USD, EUR, JPY, GBP) of the forecast rate.
n_from_bucket	This field indicates the Start Bucket Number for the given scenario.
fic_mis_date	This field indicates the As of Date for which the data being loaded is applicable.
n_fcast_rates_sys_id	The System Identifier of the assumption rule to which this data will be loaded. In case forecast name and folder are provided, then populate this field with -1.
v_folder_name	Name of the folder that holds the Forecast Rate assumption rule definition. In case the forecast sys id is provided, then populate this field with -1.
n_to_bucket	This field indicates the End Bucket Number for the given scenario.
v_xrate_method_cd	The Forecast method for exchange rate values include: Direct Input or Structured change. Use **DI** - For Direct Input or **SC** - For Structured Change
n_exchange_rate	This field indicates the Exchange rate for the Currency and given bucket Range.

STG_FCAST_IRCS

v_forecast_name	The Name of the Forecast Rate assumption rule as defined.

	The Forecast name indicates the Short Description for the Forecast Rate Sys ID as stored in the FSI_M_OBJECT_DEFINITION_TL table. In case the forecast sys id is provided, then populate this field with -1.
v_scenario_name	This field indicates the Scenario Name for which the Forecast Rate data is applicable.
v_irc_name	The IRC Name indicates the Name of Interest Rate Code .
n_interest_rate_term	This field indicates the Interest Rate Term applicable for the row of data.
v_interest_rate_term_mult	This field indicates the Interest Rate Term Multiplier for the row of data being loaded.
n_from_bucket	This field indicates the Start Bucket Number for the given scenario.
fic_mis_date	This field indicates the As of Date for which the data being loaded is applicable.
n_fcast_rates_sys_id	The System Identifier of the interest rate code forecast rate definition. In case the forecast name and folder are provided, then populate this field with -1.
v_folder_name	Name of the folder that holds the Forecast Rate assumption rule definition. In case the forecast sys id is provided, then populate this field with -1.
n_interest_rate	This field indicates the Interest Rate Change for the specified Term and for the given scenario.
n_to_bucket	This field indicates the End Bucket Number for the given scenario.

v_irc_method_cd	The Forecast method of interest rate code values include: Direct Input or Structured change. Use **DI** - For Direct Input or **SC** - For Structured Change

Forecast Rate Loader Program

The Forecast Rate Loader program updates the existing forecast rates to new forecast rates in the ALM Forecast Rate tables for Direct Input and Structured Change forecasting methods.

> **Note:** The Forecast Rate Loader can only update existing forecast rate assumption rule definitions. The initial Forecast Rate assumption rule definition and initial methods must be created through the Forecast Rates user interface within Oracle ALM.

The Forecast Rates Data Loader performs the following functions:

1. The User can load forecast rate assumptions for either a specific Forecast Rate assumption rule or multiple forecast rates assumption rules.
2. To Load a specific Forecast Rate assumption rule, the user should provide either the Forecast Rate name and a folder name as defined in Oracle ALM or the Forecast Rate System Identifier.
3. When the load parameter is to load a specific Forecast Rate assumption rule for a given As of Date, the loader checks for Forecast Name/Forecast Rate System Identifier's presence in the Object Definition Registration Table. If it's present, then the combination of Forecast Name/Forecast Rate system Identifier and As of Date is checked in each of the Forecast Rate Staging Tables one by one.
4. The data loading is done from each of the staging tables for the Direct Input and Structured change methods where the Forecast Name and As of Date combination is present.
5. When the load parameter is the Load All Option (Y), the Distinct Forecast Name from the 3 staging tables is verified for its presence in Object Definition Registration table and the loading is done for each of the Forecast Names.
6. Messages for each of the steps is written into the FSI_MESSAGE_LOG table.

After the Forecast rate loader processing is completed, the user should query the ALM Forecast Rate tables to look for the new forecast rates. Also, the user can verify the data

just loaded using the Forecast Rate Assumption UI.

Executing the Forecast Rate Data Load Procedure

The user can execute this Forecast Rate Loader from either SQL*Plus, from within a PL/SQL block or from the ICC Batch screen within OFSAAI framework.

Forecast Rate Loader – Method 1

To run the Forecast Rate Loader from SQL*Plus, login to SQL*Plus as the Schema Owner. The procedure requires 6 parameters

1. Batch Execution Identifier (batch_run_id)
2. As of Date (mis_date)
3. Forecast Rate System Identifier (pObject_Definition_ID)
4. Option for Loading All or any Specific Forecast Rate assumption rule. If the Load All option is 'N' then either the Forecast Rate Assumption rule Name Parameter with the Folder Name or Forecast Rate Sys ID should be provided else it raises an error (pLoad_all)
5. Forecast Rate assumption rule Name (pForecast_name)
6. Folder name (pFolder_Name)

The syntax for calling the procedure is:

```
fn_stg_forecast_rate_loader(batch_run_id          varchar2,
                            mis_date              varchar2,
                            pObject_Definition_ID number,
                            pLoad_all             char default 'N',
                            pForecast_name        varchar2,
                            pFolder_Name          varchar2
          p_user_id     varchar2
          p_appid       varchar2
)
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- mis_date in the format YYYYMMDD.
- pObject_Definition_ID -The Forecast Rate System Identifier in ALM
- pLoad_all indicates option for loading all forecast rates.
- pForecast_Name. This can be null i.e '' when the pLoad_all is 'Y' else provide a valid Forecast Rate assumption rule Name.

- pFolder_Name indicates the name of the Folder where the forecast rate assumption rule was defined.

- p_user_id indicates the user mapped with the application in rev_app_user_preferences. This will be used to fetch as of date from rev_app_user_preferences.This is a mandatory parameter.

- p_appid is the application name. This is a mandatory parameter.

For Example:

1. *If the user wants to Load all forecast rates assumption rules defined within a folder, say " RTSEG" then*

```
Declare
num number;
Begin
Num:= fn_stg_forecast_rate_loader('INFODOM_FORECAST_RATE_LOADER',
'20100419',
null,
'Y',
Null,
'RTSEG',
'ALMUSER1',
'ALM');
End;
```

The loading is done for all forecast rates under folder *'RTSEG' for as of Date 20100419*.

Sample Data for STG_FCAST_IRCS to Load all forecast rates defined within a folder

V_FORECAST_NAME	V_SCENARIO_NAME	V_IRC_NAME	N_INTEREST_RATE_TERM	V_INTEREST_RATE_TERM_MULT	N_FROM_BUCKET	N_INTEREST_RATE	N_TO_BUCKET	V_IRC_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	1	6.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	1	6.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	1	6.460000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	1	6.560000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	1	7.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	1	7.110000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	1	7.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	1	7.310000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	2	6.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	2	6.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	2	6.470000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	2	6.570000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	2	7.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	2	7.120000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	2	7.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	2	7.320000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	11th Dist COFI - Weekly (Code 216)	12	M	30	5.580000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	164 Monte Carlo Test	6	M	100	7.890000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	6 Month Treasury	6	M	100	5.890000		DI	4/19/2010	RTSEG	-1
TEST 2	Scenario1	3 Month Treasury	6	M	100	3.258000		DI	4/19/2010	ALMSEG	-1

Sample Data for STG_FCAST_XRATES to Load all forecast rates defined within a folder

V_FORECAST_NAME	V_SCENARIO_NAME	V_ISO_CURRENCY_CD	N_FROM_BUCKET	N_EXCHANGE_RATE	N_TO_BUCKET	V_XRATE_METHOD_CD	FIC_MIS_DATE	N_FCAST_RATES_SYS_ID	V_FOLDER_NAME
FORECAST RATE 1	Scenario1	USD	1	0.36	1	SC	4/19/2010	-1	RTSEG
FORECAST RATE 1	Scenario1	USD	2	0.25	3	SC	4/19/2010	-1	RTSEG
TEST 2	Scenario1	JPY	1	5.2	11	SC	4/19/2010	-1	RTSEG
LOADER_TEST	Scenario1	USD	1	2.5	1	DI	4/19/2010	-1	RTSEG

Sample Data for STG_FCAST_EI to Load all forecast rates defined within a folder

V_FORECAST_NAME	V_SCENARIO_NAME	V_ECONOMIC_INDICATOR_NAME	N_FROM_BUCKET	N_ECONOMIC_INDICATOR_VALUE	N_TO_BUCKET	V_EI_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	Eco 1	1	2.22	1	SC	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	Eco 1	2	1.25	3	SC	4/19/2010	RTSEG	-1
TEST 2	Scenario1	Eco 2	1	1.45	1	DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	Eco 3	1	5.28	5	SC	4/19/2010	RTSEG	-1

> **Note:** *To Load all forecast rates defined within a folder, the value of Forecast rate System identifier in the staging tables should be "-1".*

2. *If the user wants to Load a specific forecast rate assumption rule, they should provide the unique Forecast Rate System Identifier*

```
Declare
num number;
Begin
Num:= fn_stg_forecast_rate_loader('INFODOM_FORECAST_RATE_LOADER',
'20100419',
10005,
'N',
Null,
Null,
'ALMUSER1',
'ALM');
End;
```

Sample Data for STG_FCAST_IRCS to load data for specific Forecast Rate providing the Forecast Rate System Identifier

V_FORECAST_NAME	V_SCENARIO_NAME	V_IRC_NAME	N_INTEREST_RATE_TERM	V_INTEREST_RATE_TERM_MULT	N_FROM_BUCKET	N_INTEREST_RATE	N_TO_BUCKET	V_IRC_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
-1	Scenario1	New Test IRC 1	1	M	1	6.010000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	3	M	1	6.210000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	6	M	1	6.460000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	12	M	1	6.560000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	24	M	1	7.010000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	36	M	1	7.110000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	48	M	1	7.210000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	60	M	1	7.310000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	1	M	2	6.020000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	3	M	2	6.220000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	6	M	2	6.470000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	12	M	2	6.570000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	24	M	2	7.020000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	36	M	2	7.120000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	48	M	2	7.220000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	60	M	2	7.320000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	1	M	3	6.030000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	3	M	3	6.230000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	6	M	3	6.480000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	12	M	3	6.580000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	24	M	3	7.030000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	36	M	3	7.130000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	48	M	3	7.230000		DI	4/19/2010	-1	10005
-1	Scenario1	New Test IRC 1	60	M	3	7.330000		DI	4/19/2010	-1	10005
-1	Scenario1	3 Month Treasury	6	M	100	3.258000		DI	4/19/2010	-1	30008
-1	Scenario1	11th Dist COFI - Weekly (Code 216)	12	M	30	5.580000		DI	4/19/2010	-1	30256
-1	Scenario1	164 Monte Carlo Test	6	M	100	7.890000		DI	4/19/2010	-1	30256
-1	Scenario1	6 Month Treasury	6	M	100	5.890000		DI	4/19/2010	-1	30256

Sample Data for STG_FCAST_XRATES to load data for specific Forecast Rate providing the Forecast Rate System Identifier

V_FORECAST_NAME	V_SCENARIO_NAME	V_ISO_CURRENCY_CD	N_FROM_BUCKET	N_EXCHANGE_RATE	N_TO_BUCKET	V_XRATE_METHOD_CD	FIC_MIS_DATE	N_FCAST_RATES_SYS_ID	V_FOLDER_NAME
-1	Scenario1	USD	1	0.36	1	SC	4/19/2010	10005	-1
-1	Scenario1	USD	2	0.25	3	SC	4/19/2010	10005	-1
-1	Scenario1	JPY	1	5.2	11	SC	4/19/2010	30008	-1
-1	Scenario1	USD	1	2.5	1	DI	4/19/2010	30256	-1

Sample Data for STG_FCAST_EI to load data for specific Forecast Rate providing the Forecast Rate System Identifier

V_FORECAST_NAME	V_SCENARIO_NAME	V_ECONOMIC_INDICATOR_NAME	N_FROM_BUCKET	N_ECONOMIC_INDIC	N_TO_BUCKET	V_EI_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
-1	Scenario1	Eco 1	1	2.22	1	SC	4/19/2010	-1	10005
-1	Scenario1	Eco 1	2	1.25	3	SC	4/19/2010	-1	10005
-1	Scenario1	Eco 2	1	1.45	1	DI	4/19/2010	-1	30008
-1	Scenario1	Eco 3	1	5.28	5	SC	4/19/2010	-1	30256

Note: *To Load data for specific Forecast Rate providing the Forecast Rate System Identifier, the value of Forecast rate Name and Folder Name in the staging tables should be "-1".*

3. *If the user wants to Load a specific forecast rate assumption rule within the Folder providing the name of Forecast Rate as defined in ALM*

```
Declare
num number;
Begin
Num:= fn_stg_forecast_rate_loader('INFODOM_FORECAST_RATE_LOADER',
'20100419',
Null,
'N',
'LOADER_TEST',
'RTSEG',
'ALMUSER1',
'ALM');
End;
```

Sample Data for STG_FCAST_IRCS to Load a specific forecast rate within the Folder providing the name of Forecast Rate as defined in ALM

V_FORECAST_NAME	V_SCENARIO_NAME	V_IRC_NAME	N_INTEREST_RATE_TERM	V_INTEREST_RATE_TERM_MULT	N_FROM_BUCKET	N_INTEREST_RATE	N_TO_BUCKET	V_IRC_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	1	6.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	1	6.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	1	6.460000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	1	6.560000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	1	7.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	1	7.110000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	1	7.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	1	7.310000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	2	6.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	2	6.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	2	6.470000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	2	6.570000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	2	7.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	2	7.120000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	2	7.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	2	7.320000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	11th Dist COFI - Weekly (Code 216)	12	M	30	5.580000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	164 Monte Carlo Test	6	M	100	7.690000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	6 Month Treasury	6	M	100	5.690000		DI	4/19/2010	RTSEG	-1
TEST 2	Scenario1	3 Month Treasury	6	M	100	3.258000		DI	4/19/2010	ALMSEG	-1

Sample Data for STG_FCAST_XRATES to Load a specific forecast rate within the Folder providing the name of Forecast Rate as defined in ALM

V_FORECAST_NAME	V_SCENARIO_NAME	V_ISO_CURRENCY_CD	N_FROM_BUCKET	N_EXCHANGE_RATE	N_TO_BUCKET	V_XRATE_METHOD_CD	FIC_MIS_DATE	N_FCAST_RATES_SYS_ID	V_FOLDER_NAME
FORECAST RATE 1	Scenario1	USD	1	0.36	1	SC	4/19/2010	-1	RTSEG
FORECAST RATE 1	Scenario1	USD	2	0.25	3	SC	4/19/2010	-1	RTSEG
TEST 2	Scenario1	JPY	1	5.2	11	SC	4/19/2010	-1	RTSEG
LOADER_TEST	Scenario1	USD	1	2.5	1	DI	4/19/2010	-1	RTSEG

Sample Data for STG_FCAST_EI to Load a specific forecast rate within the Folder providing the name of Forecast Rate as defined in ALM

V_FORECAST_NAME	V_SCENARIO_NAME	V_ECONOMIC_INDICATOR_NAME	N_FROM_BUCKET	N_ECONOMIC_INDICATOR_VALUE	N_TO_BUCKET	V_EI_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	Eco 1	1	2.22	1	SC	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	Eco 1	2	1.25	3	SC	4/19/2010	RTSEG	-1
TEST 2	Scenario1	Eco 2	1	1.45	1	DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	Eco 3	1	5.28	5	SC	4/19/2010	RTSEG	-1

Note: *To Load a specific forecast rate assumption rule within the Folder providing the name of the Forecast Rate assumption rule as defined in ALM, the value of the Forecast rate System identifier in the staging tables should be "-1".*

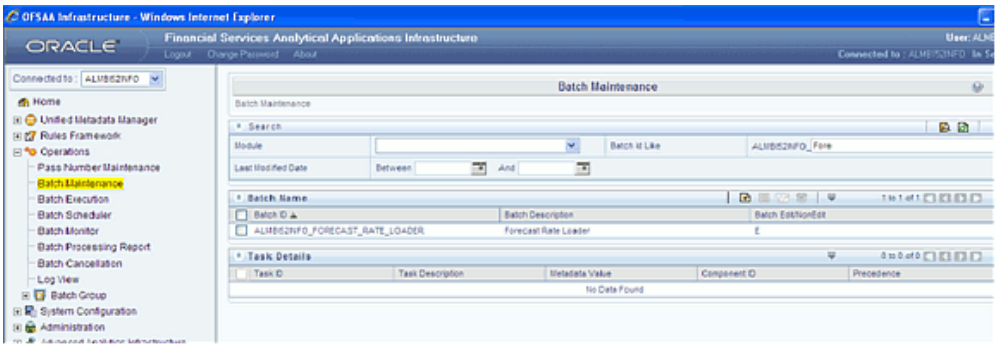
If the NUM value is 1, it indicates the load completed successfully, check the FSI_MESSAGE_LOG for more details.

Forecast Rate Loader – Method 2

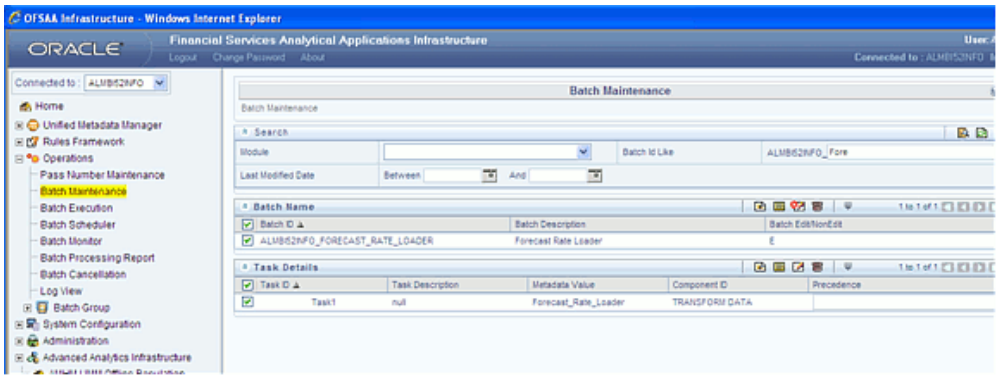
To execute Forecast Rate Loader from OFSAAI ICC framework, a seeded Batch is provided.

Steps

1. "<INFODOM>_FORECAST_RATE_LOADER" is the Batch ID and "Forecast Rate Loader" is the description of the batch.

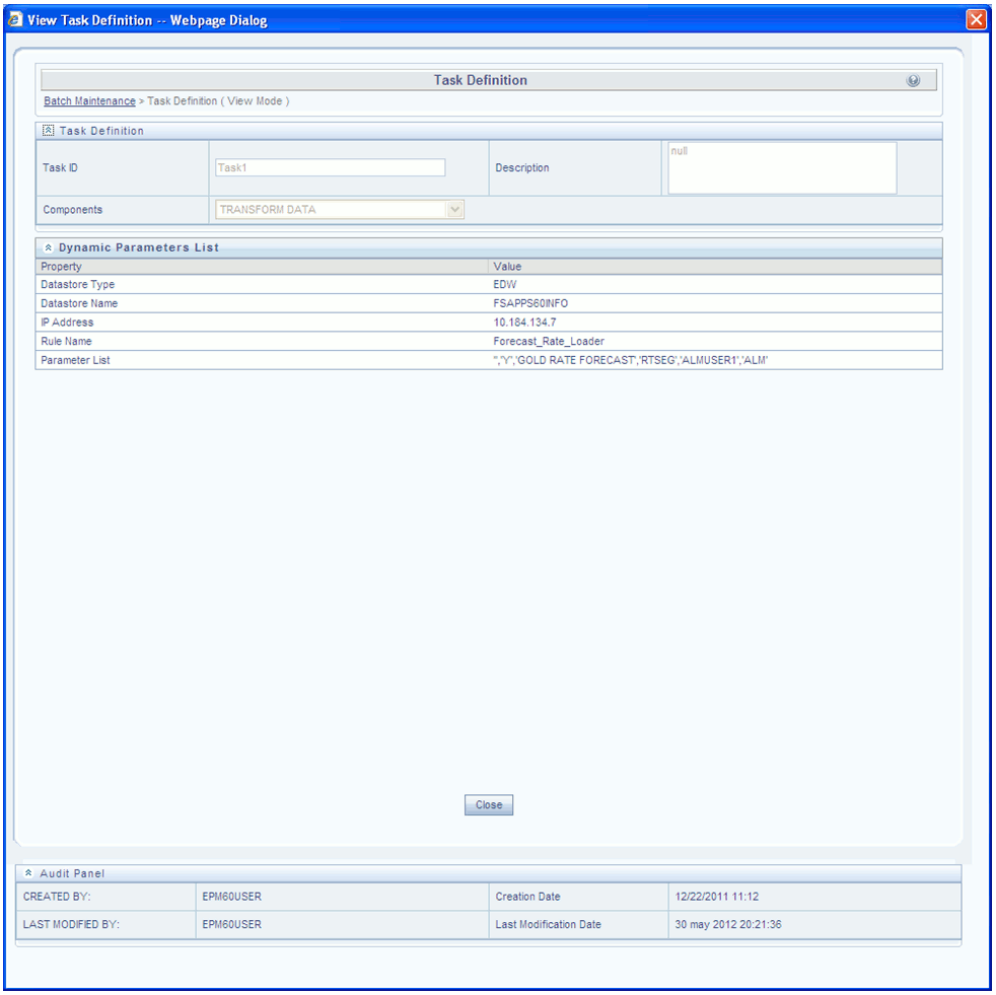


2. The batch has a single task. Edit the task.



3. If the user intends to load data for all Forecast Rates under a Folder, then provide the batch parameters as shown.

   - Datastore Type:- Select the appropriate datastore from list
   - Datastore Name:- Select the appropriate name from the list
   - IP address:- Select the IP address from the list
   - Rule Name:- **Forecast_Rate_loader**



Sample Data for STG_FCAST_IRCS to Load all forecast rates defined within a folder

V_FORECAST_NAME	V_SCENARIO_NAME	V_IRC_NAME	N_INTEREST_RATE_TERM	V_INTEREST_RATE_TERM_MULT	N_FROM_BUCKET	N_INTEREST_RATE	N_TO_BUCKET	V_IRC_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	1	6.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	1	6.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	1	6.460000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	1	6.560000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	1	7.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	1	7.110000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	1	7.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	1	7.310000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	2	6.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	2	6.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	2	6.470000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	2	6.570000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	2	7.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	2	7.120000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	2	7.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	2	7.320000		DI	4/19/2010	RTSEG	-1
LDADER_TEST	Scenario1	11th Dist COFI - Weekly (Code 216)	12	M	30	5.580000		DI	4/19/2010	RTSEG	-1
LDADER_TEST	Scenario1	164 Monte Carlo Test	6	M	100	7.890000		DI	4/19/2010	RTSEG	-1
LDADER_TEST	Scenario1	6 Month Treasury	6	M	100	5.890000		DI	4/19/2010	RTSEG	-1
TEST 2	Scenario1	3 Month Treasury	6	M	100	3.258000		DI	4/19/2010	ALMSEG	-1

Sample Data for STG_FCAST_XRATES to Load all forecast rates defined within a folder

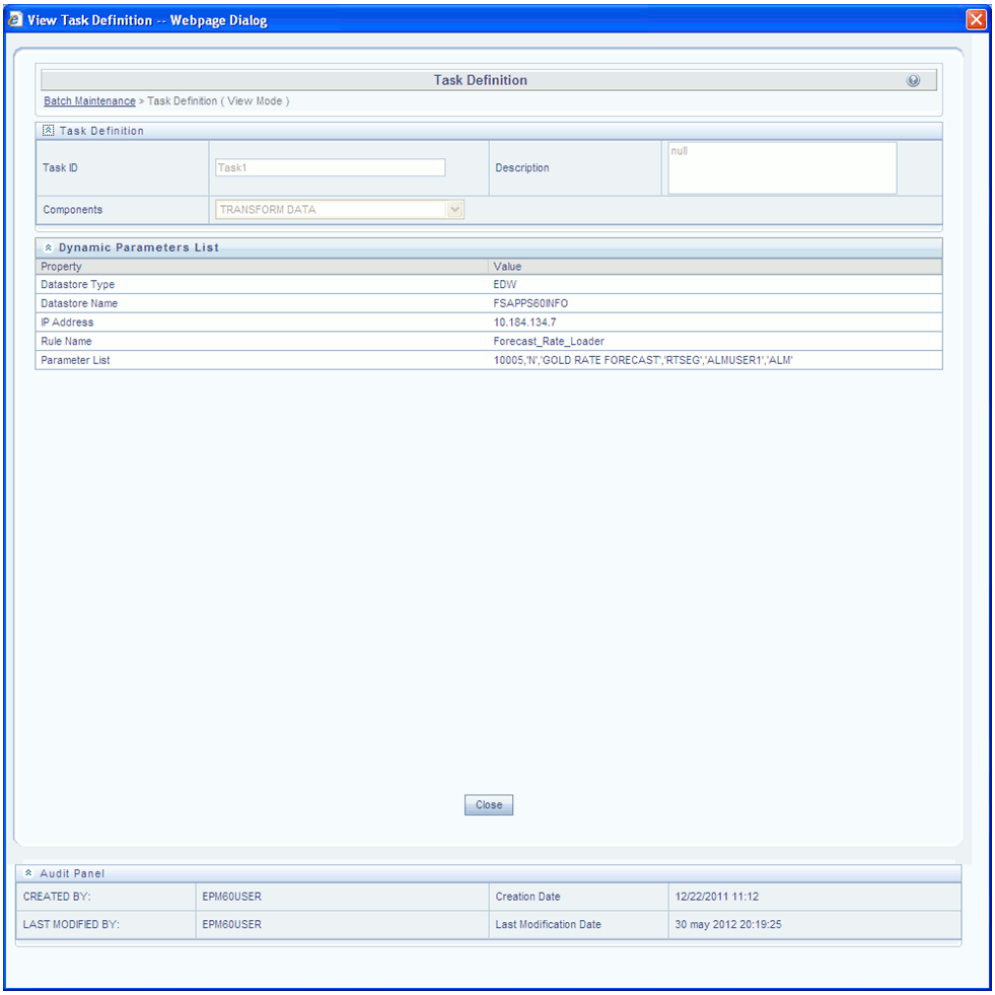
V_FORECAST_NAME	V_SCENARIO_NAME	V_ISO_CURRENCY_CD	N_FROM_BUCKET	N_EXCHANGE_RATE	N_TO_BUCKET	V_XRATE_METHOD_CD	FIC_MIS_DATE	N_FCAST_RATES_SYS_ID	V_FOLDER_NAME
FORECAST RATE 1	Scenario1	USD	1	0.36	1	SC	4/19/2010	-1	RTSEG
FORECAST RATE 1	Scenario1	USD	2	0.25	3	SC	4/19/2010	-1	RTSEG
TEST 2	Scenario1	JPY	1	5.2	11	SC	4/19/2010	-1	RTSEG
LOADER_TEST	Scenario1	USD	1	2.5	1	DI	4/19/2010	-1	RTSEG

Sample Data for STG_FCAST_EI to Load all forecast rates defined within a folder

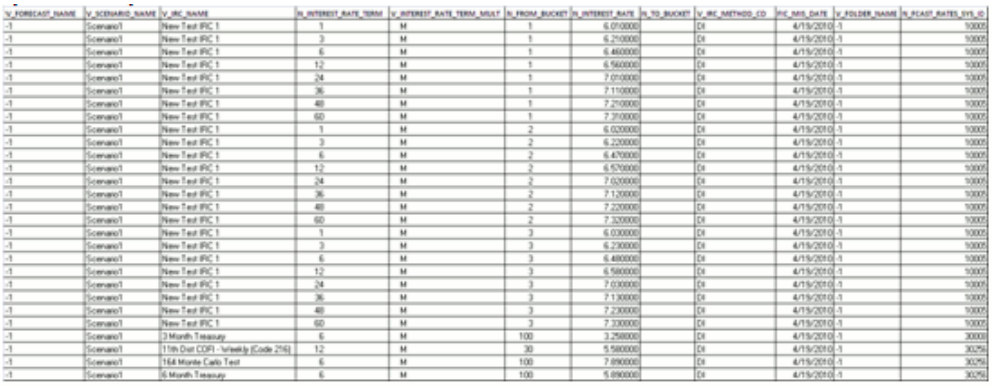
V_FORECAST_NAME	V_SCENARIO_NAME	V_ECONOMIC_INDICATOR_NAME	N_FROM_BUCKET	N_ECONOMIC_INDICATOR_VALUE	N_TO_BUCKET	V_EI_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	Eco 1	1	2.22	1	SC	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	Eco 1	2	1.25	3	SC	4/19/2010	RTSEG	-1
TEST 2	Scenario1	Eco 2	1	1.45	1	DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	Eco 3	1	5.28	5	SC	4/19/2010	RTSEG	-1

Note: *To Load all forecast rates defined within a folder, the value of Forecast rate System identifier in the staging tables should be "-1".*

4. If the user wants to load data for a specific Forecast Rate assumption rule, provide the Forecast Rate System Identifier, then define the batch parameters as shown.
   - Datastore Type:- Select the appropriate datastore from list
   - Datastore Name:- Select the appropriate name from the list
   - IP address:- Select the IP address from the list
   - Rule Name:- **Forecast_Rate_loader**


View Task Definition -- Webpage Dialog
Task Definition
Batch Maintenance > Task Definition (View Mode)
Task Definition
Task ID
Task1
Description
null
Components
TRANSFORM DATA
Dynamic Parameters List
Property
Value
Datastore Type
EDW
Datastore Name
FSAPPS60INFO
IP Address
10.184.134.7
Rule Name
Forecast_Rate_Loader
Parameter List
10005,'N','GOLD RATE FORECAST','RTSEG','ALMUSER1','ALM'
Close
Audit Panel
CREATED BY:
EPM60USER
Creation Date
12/22/2011 11:12
LAST MODIFIED BY:
EPM60USER
Last Modification Date
30 may 2012 20:19:25

Sample Data for STG_FCAST_IRCS to load data for a specific Forecast Rate assumption rule, with the Forecast Rate System Identifier already provided



Sample Data for STG_FCAST_XRATES to load data for a specific Forecast Rate assumption rule with the Forecast Rate System Identifier already provided

V_FORECAST_NAME	V_SCENARIO_NAME	V_ISO_CURRENCY_CD	N_FROM_BUCKET	N_EXCHANGE_RATE	N_TO_BUCKET	V_XRATE_METHOD_CD	FIC_MIS_DATE	N_FCAST_RATES_SYS_ID	V_FOLDER_NAME
-1	Scenario1	USD	1	0.36	1	SC	4/19/2010	10005	-1
-1	Scenario1	USD	2	0.25	3	SC	4/19/2010	10005	-1
-1	Scenario1	JPY	1	5.2	11	SC	4/19/2010	30008	-1
-1	Scenario1	USD	1	2.5	1	DI	4/19/2010	30256	-1

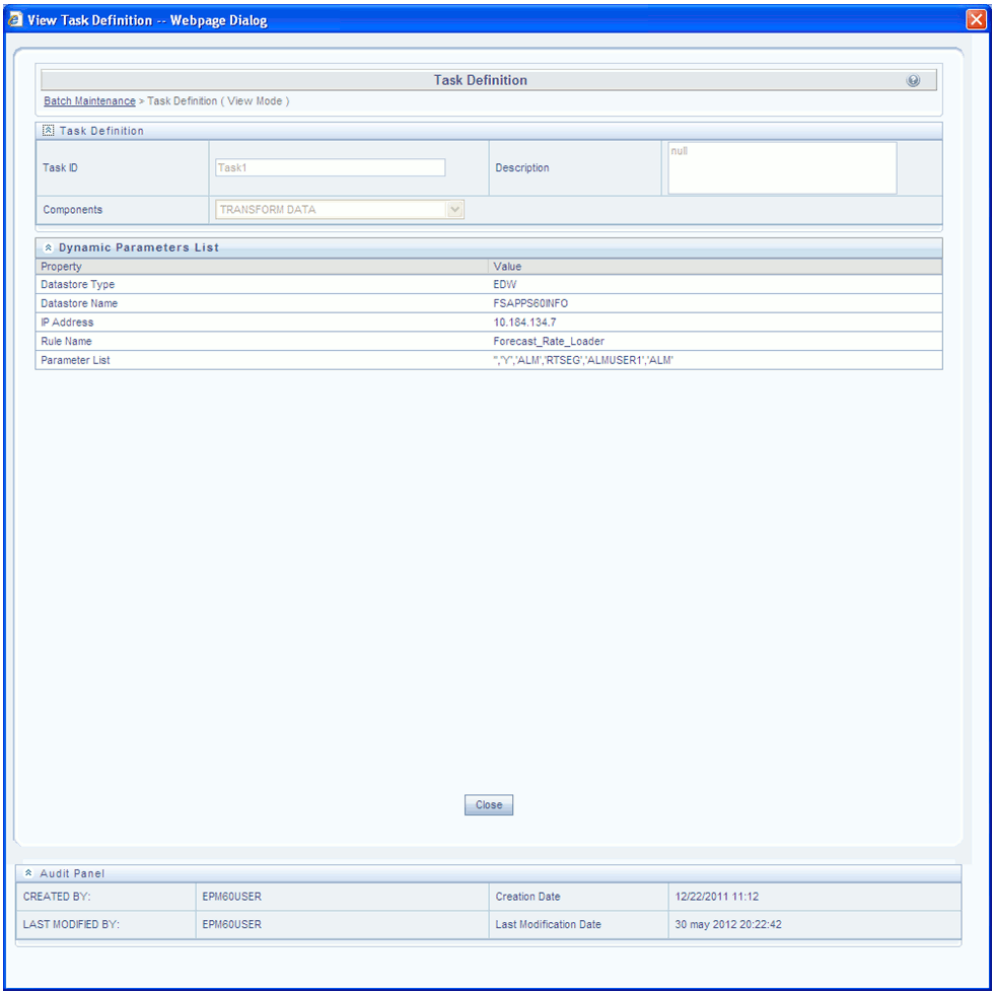
Sample Data for STG_FCAST_EI to load data for a specific Forecast Rate assumption rule with the Forecast Rate System Identifier already provided

V_FORECAST_NAME	V_SCENARIO_NAME	V_ECONOMIC_INDICATOR_NAME	N_FROM_BUCKET	N_ECONOMIC_INDIC	N_TO_BUCKET	V_EI_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
-1	Scenario1	Eco 1	1	2.22	1	SC	4/19/2010	-1	10005
-1	Scenario1	Eco 1	2	1.25	3	SC	4/19/2010	-1	10005
-1	Scenario1	Eco 2	1	1.45	1	DI	4/19/2010	-1	30008
-1	Scenario1	Eco 3	1	5.28	5	SC	4/19/2010	-1	30256

Note: *To Load data for specific Forecast Rate assumption rules, provide the Forecast Rate System Identifier and the value of Forecast rate Name and Folder Name in the staging tables should be "-1".*

5. If the user wants to load data for specific Forecast Rate assumption rules, provide the Forecast Rate Name as defined in ALM, then define the batch parameters as shown.

   - Datastore Type:- Select an appropriate datastore from list
   - Datastore Name:- Select an appropriate name from the list
   - IP address:- Select the IP address from the list
   - Rule Name:- **Forecast_Rate_loader**



Sample Data for STG_FCAST_IRCS to Load a specific forecast rate assumption rule, within the Folder, provide the name of Forecast Rate rule as defined in ALM

V_FORECAST_NAME	V_SCENARIO_NAME	V_IRC_NAME	N_INTEREST_RATE_TERM	V_INTEREST_RATE_TERM_MULT	N_FROM_BUCKET	N_INTEREST_RATE	N_TO_BUCKET	V_IRC_METHOD_CD	DIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	1	6.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	1	6.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	1	6.460000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	1	6.560000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	1	7.010000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	1	7.110000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	1	7.210000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	1	7.310000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	1	M	2	6.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	3	M	2	6.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	6	M	2	6.470000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	12	M	2	6.570000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	24	M	2	7.020000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	36	M	2	7.120000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	48	M	2	7.220000		DI	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	New Test IRC 1	60	M	2	7.320000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	11th Dist COFI - Weekly (Code 216)	12	M	30	5.580000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	164 Monte Carlo Test	6	M	100	7.890000		DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	6 Month Treasury	6	M	100	5.890000		DI	4/19/2010	RTSEG	-1
TEST 2	Scenario1	3 Month Treasury	6	M	100	3.258000		DI	4/19/2010	ALMSEG	-1

Sample Data for STG_FCAST_XRATES to Load a specific forecast rate assumption rule, within the Folder, provide the name of Forecast Rate rule as defined in ALM

V_FORECAST_NAME	V_SCENARIO_NAME	V_ISO_CURRENCY_CD	N_FROM_BUCKET	N_EXCHANGE_RATE	N_TO_BUCKET	V_XRATE_METHOD_CD	FIC_MIS_DATE	N_FCAST_RATES_SYS_ID	V_FOLDER_NAME
FORECAST RATE 1	Scenario1	USD	1	0.36	1	SC	4/19/2010	-1	RTSEG
FORECAST RATE 1	Scenario1	USD	2	0.25	3	SC	4/19/2010	-1	RTSEG
TEST 2	Scenario1	JPY	1	5.2	11	SC	4/19/2010	-1	RTSEG
LOADER_TEST	Scenario1	USD	1	2.5	1	DI	4/19/2010	-1	RTSEG

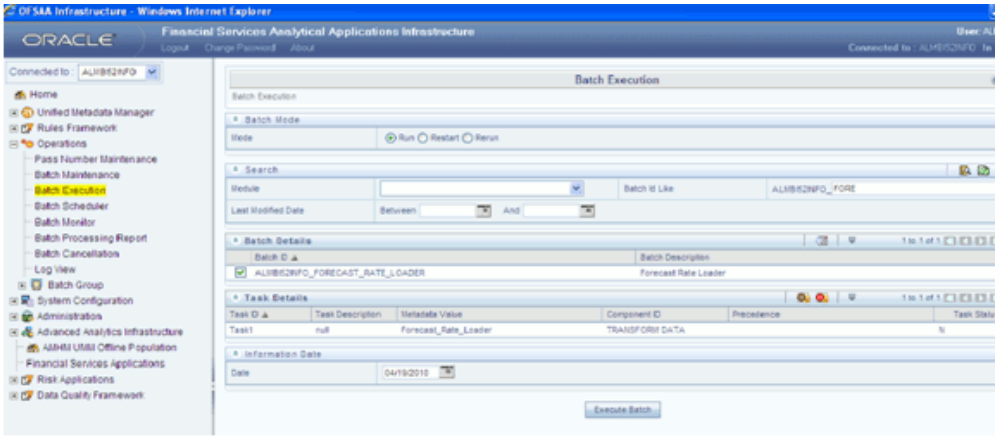
Sample Data for STG_FCAST_EI to Load a specific forecast rate assumption rule, within the Folder, provide the name of Forecast Rate rule as defined in ALM

V_FORECAST_NAME	V_SCENARIO_NAME	V_ECONOMIC_INDICATOR_NAME	N_FROM_BUCKET	N_ECONOMIC_INDICATOR_VALUE	N_TO_BUCKET	V_EI_METHOD_CD	FIC_MIS_DATE	V_FOLDER_NAME	N_FCAST_RATES_SYS_ID
FORECAST RATE 1	Scenario1	Eco 1	1	2.22	1	SC	4/19/2010	RTSEG	-1
FORECAST RATE 1	Scenario1	Eco 1	2	1.25	3	SC	4/19/2010	RTSEG	-1
TEST 2	Scenario1	Eco 2	1	1.45	1	DI	4/19/2010	RTSEG	-1
LOADER_TEST	Scenario1	Eco 3	1	5.28	5	SC	4/19/2010	RTSEG	-1

Note: *To Load a specific forecast rate assumption rule within the Folder, provide the name of the Forecast Rate rule as defined in ALM. The Forecast rate System identifier in the staging tables should be "-1".*

6. Save the Batch.

7. Execute the Batch for the required As of Date.



Exception Messages

The Forecast Rate Data Loader can have the following exceptions:

Exception 1: Error. While fetching the Object Definition ID from Object Registration Table

This exception occurs if the forecast rate assumption rule name is not present in the FSI_M_OBJECT_DEFINTION_TL table short_desc column.

Exception 2: Error. More than one Forecast Sys ID is present.

This exception occurs when there is more than one Forecast Sys ID present for the given forecast rate assumption rule name.

Exception 3: Error. Forecast Rate assumption rule Name and As of Date combination do not exist in the Staging Table.

This exception occurs when the Forecast Rate assumption rule Name and as of date combination do not exist in the Staging Table.

Prepayment Rate Data Loader

The Prepayment Rate Data Loader procedure populates prepayment model rates (used in ALM and FTP) into the OFSAA metadata tables from the corresponding staging tables. Prepayment model assumptions are defined within the Prepayment Model Assumptions User Interfaces in OFSAA ALM and FTP applications. This data loader program can be used to update the prepayment model rates on a periodic basis. After loading the prepayment rates, users can view the latest data in the Prepayment Model assumptions UI.

The following topics are covered in this section:

- Tables related to the Prepayment rate data loader
- Prepayment Rate Data Load Procedure
- Executing the Prepayment Rate Data Loader
- Exception Messages

Prepayment Rate Loader Tables

The following are the tables used by the loader:

- FSI_PPMT_MODEL_HYPERCUBE – This table contains rates defined for different Prepayment Dimensions present in FSI_PPMT_MODEL_HYPERCUBE_MAP table.
- STG_PPMT_MDL_HYPERCUBE – contains prepayment rates for the selected prepayment dimensions.

Prepayment Rate Data Loader

The Prepayment Rate Data Loader program populates the OFSAA Prepayment Model tables with the values from the staging table. The procedure will load prepayment assumption data for all Prepayment models that are present in the staging table. The program assumes that the prepayment model definitions have already been defined

using OFSAA Prepayment Model assumptions UIs before loading prepayment model rates.

The program performs the following functions:

1. The Data Loader accepts the AS_OF_DATE as a parameter, that is, date to load all prepayment rates from the Staging table into the OFSAA metadata table for the specific as of date.

2. The program performs certain checks to determine if:

   - The prepayment model dimensions present in staging are the same as those present in the OFSAA Prepayment Model metadata tables.

   - The members of each of the dimensions present in staging are same as those present in the metadata tables.

   - The number of records present in the STG_PPMT_MDL_HYPERCUBE table for a Prepayment model is less than or equal to the maximum number of records that are allowed which is determined by multiplying the number of buckets per dimension of the prepayment model.

     Example

PPMT_MDL_SYS_ID	DIMENSION_ID	NUMBER_OF_BUCKETS
20100405	8	2
20100405	4	3

     Then the maximum number of records = number of buckets of dimension 8 * number of buckets of dimension 4

     That is, maximum number of records = 2 * 3

     Therefore, maximum number of records = 6 records

     Check is made by Prepayment Rate Data Loader whether the number of records present in STG_PPMT_MDL_HYPERCUBE table for a Prepayment model 20100405 is less than or equal to 6 or not.

   If the above quality checks are satisfied, then the rates present in the Staging table are updated to the OFSAA prepayment model metadata table.

3. Any error messages are logged in the FSI_MESSAGE_LOG table and can be viewed in OFSAAI Log Viewer UI.

After the Prepayment Rate loader is completed, you should query the FSI_PPMT_MODEL_HYPERCUBE table to look for the new rates. Also, you can verify the data using the Prepayment Model Assumption UI.

Executing the Prepayment Model Data Loader

You can execute this function within a PL/SQL block or from an ICC Batch screen within OFSAAI framework.

To run the function from SQL*Plus, login to SQL*Plus as the Schema Owner. The loader requires 2 parameters

- Batch Execution Name
- As Of Date

fn_PPMT_RATE_LOADER(batch_run_id IN VARCHAR2, as_of_date IN VARCHAR2)

BATCH_RUN_ID is any string to identify the executed batch.

As_of_Date is the execution date in the format YYYYMMDD.

For Example:

```
Declare
      num number;
Begin
      Num:= fn_PPMT_RATE_LOADER('INFODOM_20100405','20100405');
End;
```

The loader is executed for the given as of date. If the return value (NUM) is 1, this indicates the load completed successfully. Check the FSI_MESSAGE_LOG for more details.

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as PPMTMODELRATELOADER and specify the following parameters for the task:

Datastore Type:- Select appropriate datastore from list

Datastore Name:- Select appropriate name from the list

IP address:- Select the IP address from the list

Rule Name:- **ppmt_rate_loader**

Parameter List: None

Exception Messages

The Prepayment Model Rate Loader can have the following exceptions:

Exception 1: Error while fetching the Object Definition ID from Object Definition Table.

This exception occurs if the prepayment model name is not present in the FSI_M_OBJECT_DEFINTION_TL table.

Exception 2: Error. More than one prepayment model sys ID is present for the given definition.

This exception occurs when there is more than one Prepayment Model System ID present for the prepayment model name in staging.

Exception 3: Error. Data is present in additional dimension ID column than those defined in FSI_M_PPMT_MODEL.

This exception occurs if rates are specified in staging for the dimensions that are not part of the Prepayment Model definition.

Exception 4: The value in the Dimension ID column is not matching with the value present in the corresponding column in metadata table.

This exception occurs if rates are specified in staging for the dimension members that are not part of the Prepayment Model definition.

Exception 5: The number of records for the staging table for a given Prepayment Model Name is more than those calculated by multiplying the number of buckets in FSI_M_PPMT_MODEL table for the given model name.

This exception occurs if there are excess records in staging compared to OFSAA metadata tables for the given prepayment model.

Stage Instrument Table Loader

Data in staging instrument tables are moved into respective OFSAA processing instrument tables using OFSAAI T2T component. After loading the data, users can view the loaded data by querying the processing instrument tables.

The following topics are covered in this section:

- Stage Tables
- Populating Stage tables
- Mapping between staging and OFSAA processing tables
- Populating Account Dimension
- Executing T2T data movement tasks
- Re-loading records

Stage Tables

Following are examples of some of the various application staging instrument tables:

- STG_LOAN_CONTRACTS – holds contract information related to various loan products including mortgages.
- STG_TD_CONTRACTS – holds contract information related to term deposit products.
- STG_CASA – holds information related to Checking and Savings Accounts.
- STG_OD_ACCOUNTS – holds information related to over-draft accounts.
- STG_CARDS – holds information related to credit card accounts.
- STG_LEASES – holds contract information related to leasing products.
- STG_ANNUITY_CONTRACTS – holds contract information related to annuity contracts.
- STG_INVESTMENTS – holds information related to investment products like bond, equities etc.
- STG_MM_CONTRACTS – holds contract information related to short term investments in money market securities.
- STG_BORROWINGS – holds contract information related to various inter-bank borrowings.
- STG_FX_CONTRACTS – holds contract information related to FX products like FX Spot, FX Forward etc. Leg level details, if any, are stored in various leg-specific columns within the table.
- STG_SWAPS_CONTRACTS – holds contract information related to various types of swaps. Leg level details, if any, are stored in various leg-specific columns within the table.
- STG_OPTION_CONTRACTS – holds contract information related to various types of options. Leg level details, if any, are stored in various leg-specific columns within the table.
- STG_FUTURES – holds contract information related to interest rate forwards and all types of futures. Leg level details, if any, are stored in various leg-specific columns within the table.

Populating Stage Tables

Data can be loaded into staging tables through F2T component of OFSAAI. After data is loaded, check for data quality within the staging tables, before moving into OFSAA processing tables. Data quality checks can include:

- Number of records between external system and staging instrument tables.
- Valid list of values in code columns of staging.
- Valid list of values in dimension columns like product, organization unit, general ledger etc. These members should be present in the respective dimension tables.
- Valid values for other significant columns of staging tables.

Mapping To OFSAA Processing Tables

Following are examples of some of the pre-defined application T2T mappings between the above staging tables and processing tables:

- T2T_LOAN_CONTRACTS – for loading data from STG_LOAN_CONTRACTS to FSI_D_LOAN_CONTRACTS.
- T2T_MORTGAGES – for loading data from STG_LOAN_CONTRACTS to FSI_D_MORTGAGES.
- T2T_CASA – for loading data from STG_CASA to FSI_D_CASA.
- T2T_CARDS – for loading data from STG_CARDS to FSI_D_CREDIT_CARDS.
- T2T_TD_CONTRACTS – for loading data from STG_TD_CONTRACTS to FSI_D_TERM_DEPOSITS.
- T2T_ANNUITY_CONTRACTS – for loading data from STG_ANNUITY_CONTRACTS to FSI_D_ANNUITY_CONTRACTS.
- T2T_BORROWINGS – for loading data from STG_BORROWINGS to FSI_D_BORROWINGS.
- T2T_FORWARD_CONTRACTS – for loading data from STG_FUTURES to FSI_D_FORWARD_RATE_AGMTS.
- T2T_FUTURE_CONTRACTS – for loading data from STG_FUTURES to FSI_D_FUTURES.
- T2T_FX_CONTRACTS – for loading data from STG_FX_CONTRACTS to FSI_D_FX_CONTRACTS.
- T2T_INVESTMENTS – for loading data from STG_INVESTMENTS to FSI_D_INVESTMENTS.
- T2T_LEASES_CONTRACTS – for loading data from STG_LEASES_CONTRACTS to FSI_D_LEASES.

- T2T_MM_CONTRACTS – for loading data from STG_MM_CONTRACTS table to FSI_D_MM_CONTRACTS.
- T2T_OPTION_CONTRACTS – for loading data from STG_OPTION_CONTRACTS to FSI_D_OPTION_CONTRACTS.
- T2T_SWAP_CONTRACTS – for loading data from STG_SWAPS_CONTRACTS to FSI_D_SWAPS.
- T2T_OD_ACCOUNTS – for loading data from STG_OD_ACCOUNTS to FSI_D_CREDIT_LINES.

You can view the Database Extract definitions by performing the following steps:

> **Note:** The *Data Management Tools* and *Data Ingestion* were previously known as *Data Integrator Framework* and *Warehouse Designer* respectively. These new terminologies are applicable only for OFSAAI versions 7.3.2.3.0 and above.

- Navigate to *Unified Metadata Manager > Data Management Tools > Data Ingestion > Database Extracts* section.
- In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
- Expand the required T2T definition to view the Database Extract definitions.

You can view the Source - Target mapping definitions by performing the following steps:

> **Note:** The *Data Management Tools* and *Data Ingestion* were previously known as *Data Integrator Framework* and *Warehouse Designer* respectively. These new terminologies are applicable only for OFSAAI versions 7.3.2.3.0 and above.

- Navigate to *Unified Metadata Manager > Data Management Tools > Data Ingestion > Database Extracts* section.
- In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
- Expand the required T2T definition to view the extract definition.
- Click the required Database Extract definition.

  The selected Database Extract definition details are displayed with the available Source - Target mappings under the *Source - Target Mappings* grid.

Note: Staging instrument tables contain alphanumeric display codes for various IDENTIFIER and CODE columns. T2T mapping looks up in respective dimension tables for fetching an equivalent numeric ID and CODE corresponding to the alphanumeric display code. Hence, these dimension tables should be populated with the alphanumeric display code before executing any data movement tasks.

Populating Accounts Dimension

Account Number is an alphanumeric unique identifier within each staging instrument tables. ID_NUMBER is a numeric unique identifier within processing instrument tables. Hence, there is a need to generate a numeric surrogate key for each of the account number. This information is stored in DIM_ACCOUNT table.

Function **fn_popDimAccount** is a function to populate numeric surrogate key for each account number. The function performs the following:

- In case surrogate key generation is required, then it uses a sequence to populate DIM_ACCOUNT table.
- In case surrogate key generation is not required, then it expects that the account number to be numeric and populates DIM_ACCOUNT with that information.

Create a new Batch with the Task and specify the following parameters for the task to populate DIM_ACCOUNT table:

- Datastore Type:- Select appropriate datastore from the drop down list.
- Datastore Name: - Select appropriate name from the list. Generally it is the Infodom name.
- IP address:- Select the IP address from the list.
- Rule Name:- **fn_popDimAccount**
- Parameter List:
  - Surrogate Key Required Flag – Y or N

  Batch run ID and As Of Date are passed internally by the batch to the Data Transformation task.

Executing T2T Data Movement Tasks

Before executing T2T data movement tasks, user should ensure that all the dimension tables that are required for instruments data are loaded. The following are some of the mandatory dimensions:

- DIM_ACCOUNTS
- DIM_PRODUCTS_B
- DIM_GENERAL_LEDGER_B
- DIM_COMMON_COA_B
- DIM_ORG_UNIT_B

Create a new Batch with the Task and specify the following parameters for the task for loading Historical Exchange Rates:

- Datastore Type:- Select appropriate datastore from the drop down list.
- Datastore Name: - Select appropriate name from the list. Generally it is the Infodom name.
- IP address:- Select the IP address from the list.
- Rule Name:- Select the appropriate T2T name from the above list.
- Parameter List: No Parameter is passed. The only parameter is the As of Date Selection while execution.

Check T2T component logs and batch messages for checking the status of load.

T2T component can fail because of following cases:

- Unique constraint error – Target table may already contain the primary keys that are part of the staging tables.
- NOT NULL constraint error – do not have values for NOT NULL columns in the target table.

Re-Load Of Instrument Data

T2T component can only perform "Insert" operations. In case user needs to perform updates, previously loaded records should be deleted before loading the current records.

Function **fn_deleteFusionTables** is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function needs FSI_DELETE_TABLES_SETUP to be configured. Configure the below table for all columns that need to be part of the join between Stage table and Equivalent table.

Create a new Batch with the Task and specify the following parameters for the task to delete existing records:

- Datastore Type: - Select appropriate datastore from the drop down list.
- Datastore Name: - Select appropriate name from the list. Generally it is the Infodom name.
- IP address:- Select the IP address from the list.
- Rule Name:- **fn_deleteFusionTables**
- Parameter List:
  - 'Table to be deleted'

  Batch run ID and As Of Date are passed internally by the batch to the Data Transformation task.

Sample record for **FSI_DELETE_TABLES_SETUP** table is given below:

Column Name	Column Description	Sample Value
STAGE_TABLE_NAME	Stores the source table name for forming the join statement	STG_LOAN_CONTRACTS
STAGE_COLUMN_NAME	Stores the source column name for forming the join statement	V_ACCOUNT_NUMBER
FUSION_TABLE_NAME	Stores the target table name for forming the join statement	FSI_D_LOAN_CONTRACTS
FUSION_COLUMN_NAME	Stores the target column name for forming the join statement	ACCOUNT_NUMBER

Note: Insert rows in FSI_DELETE_TABLES_SETUP for all columns that can be used to join the stage with the equivalent table. In case if the join requires other dimension or code tables, a view can be created joining the source table with the respective code tables and that view can be part of the above setup table.

Transaction Summary Table Loader

Data in staging transaction summary tables are moved into respective OFSAA processing transaction summary tables using OFSAAI T2T component. After loading

the data, users can view the loaded data by querying the processing transaction tables.

The following topics are covered in this section:

- Stage Tables
- Populating Stage tables
- Mapping between staging and OFSAA processing tables
- Dependencies
- Executing T2T data movement tasks
- Re-loading records

Stage Tables

Following are examples of various application staging transaction summary tables:

- STG_LOAN_CONTRACT_TXNS_SUMMARY – holds transaction summary information related to the loan contracts that are present in staging instrument table for loan contracts, that is STG_LOAN_CONTRACTS.
- STG_CARDS_TXNS_SUMMARY – holds transaction summary information related to the credit cards present that are present in staging instrument table for credit cards, that is STG_CARDS.
- STG_CASA_TXNS_SUMMARY – holds transaction summary information related to the checking and saving accounts that are present in staging instrument table for CASA, that is STG_CASA.
- STG_MERCHANT_CARD_TXNS_SUMMARY – holds transaction summary information related to the merchant cards that are present in staging instrument table for merchant cards, that is STG_MERCHANT_CARDS.
- STG_OTHER_SERVICE_TXNS_SUMMARY – holds transaction summary information related to other services that are present in staging instrument table for other services, that is STG_OTHER_SERVICES.
- STG_TERMDEPOSITS_TXNS_SUMMARY – holds transaction summary information related to the term deposits that are present in staging instrument table for term deposits, that is STG_TD_CONTRACTS.
- STG_TRUSTS_TXNS_SUMMARY – holds transaction summary information related to the trust accounts that are present in staging instrument table for trusts, that is STG_TRUSTS.

Populating Stage Tables

Data can be loaded into staging tables through F2T component of OFSAAI. After data is loaded, check for data quality within the staging tables, before moving into OFSAA processing tables. Data quality checks can include:

- Number of records between external system and staging transaction summary tables.
- Valid list of values in code columns of staging.
- Valid list of values in dimension columns like product, organization unit, general ledger etc. These members should be present in the respective dimension tables.
- Valid list of values in dimension columns like product, organization unit, general ledger etc. These members should be present in the respective dimension tables.
- Valid values for other significant columns of staging tables.

Mapping To OFSAA Processing Tables

Following are examples of the pre-defined T2T mappings between the above application staging tables and processing tables:

- T2T_STG_CARDS_TXNS_SUMMARY – for loading data from STG_CARDS_TXNS_SUMMARY to FSI_D_CREDIT_CARDS_TXNS.
- T2T_STG_CASA_TXNS_SUMMARY – for loading data from STG_CASA_TXNS_SUMMARY to FSI_D_CASA_TXNS.
- T2T_LOAN_CONTRACT_TXNS_SUMMARY – for loading data from STG_LOAN_CONTRACT_TXNS_SUMMARY to FSI_D_LOAN_CONTRACTS_TXNS.
- T2T_STG_MERCHANT_CARD_TXNS_SUMMARY – for loading data from STG_MERCHANT_CARD_TXNS_SUMMARY to FSI_D_MERCHANT_CARDS_TXNS.
- T2T_STG_OTHER_SERVICE_TXNS_SUMMARY – for loading data from STG_OTHER_SERVICE_TXNS_SUMMARY to FSI_D_OTHER_SERVICES_TXNS.
- T2T_STG_TERMDEPOSITS_TXNS_SUMMARY – for loading data from STG_TERMDEPOSITS_TXNS_SUMMARY to FSI_D_TERM_DEPOSITS_TXNS.
- T2T_STG_TRUSTS_TXNS_SUMMARY – for loading data from STG_TRUSTS_TXNS_SUMMARY to FSI_D_TRUSTS_TXNS.

You can view the Database Extract definitions by performing the following steps:

> **Note:** The *Data Management Tools* and *Data Ingestion* were previously known as *Data Integrator Framework* and *Warehouse Designer* respectively. These new terminologies are applicable only for OFSAAI versions 7.3.2.3.0 and above.

- Navigate to *Unified Metadata Manager* > *Data Management Tools* > *Data Ingestion* > *Database Extracts* section.
- In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
- Expand the required T2T definition to view the Database Extract definitions.

You can view the Source - Target mapping definitions by performing the following steps:

> **Note:** The *Data Management Tools* and *Data Ingestion* were previously known as *Data Integrator Framework* and *Warehouse Designer* respectively. These new terminologies are applicable only for OFSAAI versions 7.3.2.3.0 and above.

- Navigate to *Unified Metadata Manager* > *Data Management Tools* > *Data Ingestion* > *Database Extracts* section.
- In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
- Expand the required T2T definition to view the extract definition.
- Click the required Database Extract definition.

  The selected Database Extract definition details are displayed with the available Source - Target mappings under the *Source - Target Mappings* grid.

  > **Note:** Staging transaction summary tables contain alphanumeric display codes for various IDENTIFIER and CODE columns. T2T mapping looks up in respective dimension tables for fetching an equivalent numeric ID and CODE corresponding to the alphanumeric display code. Hence, these dimension tables should be populated with the alphanumeric display code before executing any data movement tasks.

Dependencies

- Instrument tables should be loaded before loading the transaction summary information related to those instruments.
- Account Number is an alphanumeric unique identifier within each staging transaction summary tables. ID_NUMBER is a numeric unique identifier within processing transaction summary tables. Hence, there is a need to look up into a DIM_ACCOUNT dimension table for a numeric surrogate key for each of the alphanumeric account number. This dimension table DIM_ACCOUNT will be populated as part of the process that loads instrument tables. For more information on loading instrument tables, see Loading Instrument Table Data, page 4-49.
- Before executing T2T data movement tasks, user should ensure that all the dimension tables that are required for instruments data are loaded. The following are some of the mandatory dimensions:
  - DIM_ACCOUNTS
  - DIM_PRODUCTS_B
  - DIM_GENERAL_LEDGER_B
  - DIM_COMMON_COA_B
  - DIM_ORG_UNIT_B

Executing T2T Data Movement Tasks

Create a new Batch with the Task and specify the following parameters for the task for loading Historical Exchange Rates:

- Datastore Type: - Select appropriate datastore from the drop down list.
- Datastore Name: - Select appropriate name from the list. Generally it is the Infodom name.
- IP address:- Select the IP address from the list.
- Rule Name: - Select the appropriate T2T name from the above list.
- Parameter List: - No Parameter is passed. The only parameter is the As of Date Selection while execution.

Check T2T component logs and batch messages for checking the status of load.

T2T component can fail because of following cases:

- Unique constraint error – Target table may already contain the primary keys that are part of the staging tables.
- NOT NULL constraint error – Staging table do not have values for mandatory columns of the target table.

Re-Load Of Transaction Summary Data

T2T component can only perform "Insert" operations. In case user needs to perform updates, previously loaded records should be deleted before loading the current records.

Function **fn_deleteFusionTables** is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function needs FSI_DELETE_TABLES_SETUP to be configured. Configure the below table for all columns that need to be part of the join between Stage table and Equivalent table.

Create a new Batch with the Task and specify the following parameters for the task to delete existing records:

- Datastore Type: - Select appropriate datastore from the drop down list.
- Datastore Name: - Select appropriate name from the list. Generally it is the Infodom name.
- IP address: - Select the IP address from the list.
- Rule Name:- **fn_deleteFusionTables**
- Parameter List:
  - 'Table to be deleted'

  Batch run ID and As Of Date are passed internally by the batch to the Data Transformation task.

Sample record for **FSI_DELETE_TABLES_SETUP** table is given below:

Column Name	Column Description	Sample Value
STAGE_TABLE_NAME	Stores the source table name for forming the join statement	STG_LOAN_CONTRACTS
STAGE_COLUMN_NAME	Stores the source column name for forming the join statement	V_ACCOUNT_NUMBER

Column Name	Column Description	Sample Value
FUSION_TABLE_NAME	Stores the target table name for forming the join statement	FSI_D_LOAN_CONTRACTS
FUSION_COLUMN_NAME	Stores the target column name for forming the join statement	ACCOUNT_NUMBER

Note: Insert rows in FSI_DELETE_TABLES_SETUP for all columns that can be used to join the stage with the equivalent table. In case if the join requires other dimension or code tables, a view can be created joining the source table with the respective code tables and that view can be part of the above setup table.

Ledger Data Loader

The LEDGER_STAT load utility is an Oracle stored procedure used to load your ledger data into the Oracle Financial Services Analytical Applications (OFSAA) LEDGER_STAT table. The following topics are included in this section:

- Features of the load procedure
- Overview of the load procedure
- Setup for the LEDGER_STAT load utility
- Executing LEDGER_STAT load procedure

Features of the load procedure

The LEDGER_STAT load utility is the only supported method for loading your ledger data into the LEDGER_STAT table. The LEDGER_STAT load utility offers the following features:

- You can load ledger data for one month or for a range of months.
- You can also load ledger data based on calendar as-of-dates.
- A month can be undone individually, using the Ledger Load Undo process. You can do this even though the month to be undone is included in a multiple-month load.

- You can update columns in existing LEDGER_STAT rows using either the additive or replacement functionality.

- You can bypass the upsert logic and insert all the rows from the load table using the INSERT_ONLY mode. This functionality can be used either for first-time loads or to reload for all months with each load.

Overview of the Load Process

There are three types of load tables that can be used for loading ledger data.

- Type I (FISCAL_ONE_MONTH) – Load table contains ONE_MONTH column for storing data corresponding to one of the twelve fiscal months.

- Type II (FISCAL_RANGE) – Load table contains M1 to M12 columns for storing data corresponding to twelve fiscal months.

- Type III (CALENDAR_MONTHS) – Load table contains AS_OF_DATE for storing data corresponding to an as-of-date. While Type II table contains ledger data across fiscal months in a single row, Type III contains the same information in multiple rows. Type III supports calendar dates and data can be for one or multiple dates.

ASCII Ledger data is loaded into any of the above staging or load tables using F2T component of OFSAAI framework. This component can be used for loading any flat file data into tables. For more information on how to load data using F2T, see *OFSAAI User Guide*.

LEDGER_STAT load utility is a PL/SQL procedure and loads data from the above staging tables into LEDGER_STAT table, based on the configuration. Runtime parameters, such as the name of the load table, which all columns to load, ADD or REPLACE update functionality, and whether or not to create offset records are passed as parameters to the procedure and these are inserted into the Load Batch table (FSI_LS_LOAD_BATCH).

The procedure is implemented as an Oracle PL/SQL stored procedure so it can be invoked from SQL*Plus or Batch execution screen within OFSAAI ICC framework component. Input parameters are read from the batch/parameter table and validated for correctness, completeness and consistency before the load begins. Parameter errors are written to a Message column in the batch/parameter table and FSI_MESSAGE_LOG table. Runtime statistics are written to the batch/parameter record following completion of the load for that record.

> **Note:** For supporting loading LEDGER_STAT from Type III staging table, a global temporary table (GTT) is created within database. Data is moved from global temporary table into LEDGER_STAT table.

Limitations

The following are the limitations.

- **Load Table Rows Must Be Unique**

  A restriction imposed by the use of bulk SQL (as opposed to row-by-row) processing is that all the rows in the load table(s) must be unique. This means that there is one row in the load table for one row in LEDGER_STAT. A unique index is created on each load table to enforce this uniqueness and provide acceptable performance.

- **Defining Financial Elements in AMHM**

  Occasionally, your load table may contain dimension member values for one or more dimensions that are not defined in AMHM. The LEDGER_STAT load procedure loads these rows anyway, except for the rows containing undefined or incompletely defined FINANCIAL_ELEM_ID values.

  Any new values for FINANCIAL_ELEM_ID must first be defined in AMHM before running the load. Specifically, the load procedure needs the AGGREGATE_METHOD value for each FINANCIAL_ELEM_ID value so that the YTD columns in LEDGER_STAT can be computed using the appropriate method.

Setup for the LEDGER_STAT load utility

Setting up and Executing a Type III (or Type 3) Ledger Stat Load Using STG_GL_DATA

The Type 3 load takes data from STG_GL_DATA and transfers it into the LEDGER_STAT table.

Steps to follow to setup and run a Type III Ledger Stat Load:

Step 1: Populate STG_GL_DATA

The following columns in STG_GL_DATA must be populated with valid values:

V_GL_CODE	General Ledger "Code" value.
FIC_MIS_DATE	This field indicates the current period As of Date applicable to the data being loaded.
V_ORG_UNIT_CODE	Org Unit "Code" value.

V_SCENARIO_CODE	Populate with a value from the CONSOLIDATION_DISPLAY_CODE column from the FSI_CONSOLIDATION_CD table (ex. ACTUAL, BUDGET).
V_CCY_CODE	ISO Currency Code from FSI_CURRENCIES (ex. USD)
V_PROD_CODE	Product "Code" value.
V_FINANCIAL_ELEMENT_CODE	Populate with a value from the FINANCIAL_ELEM_CODE column from the DIM_FINANCIAL_ELEMENTS_B table (ex. ENDBAL, AVGBAL).
V_COMMON_COA_CODE	Common COA "Code" value.
N_AMOUNT_LCY	Balance

The following columns in STG_GL_DATA must be populated because they are defined as NOT NULL but can be defaulted to the value of your choice because they are not used: V_LV_CODE

V_BRANCH_CODE

F_CONSOLIDATION_FLAG

V_GAAP_CODE

Step 2: Verify data exists in the view STG_GL_DATA_V

The following SQL statement is used to populate this view:

```
SELECT v_data_origin DS,
   f_consolidation_flag ACCUM_TYPE,
   fcc.consolidation_cd CONSOLIDAT,
   v_ccy_code ISOCRNCYCD,
   dfeb.financial_elem_id FINANC_ID,
   doub.org_unit_id ORG_ID,
   dglb.gl_account_id GL_ACCT_ID,
   dccb.common_coa_id CMN_COA_ID,
   dpb.product_id PRDCT_ID,
   fic_mis_date AS_OF_DATE,
   n_amount_lcy VALUE,
   0 baltypecd
FROM STG_GL_DATA SGD,
DIM_GENERAL_LEDGER_B DGLB,
DIM_ORG_UNIT_B DOUB,
DIM_PRODUCTS_B DPB,
DIM_FINANCIAL_ELEMENTS_B DFEB,
DIM_COMMON_COA_B DCCB,
FSI_CURRENCIES FC,
FSI_CONSOLIDATION_CD FCC
WHERE NVL(n_amount_lcy, 0) <> 0
AND SGD.V_GL_CODE = DGLB.GL_ACCOUNT_CODE
AND SGD.V_ORG_UNIT_CODE = DOUB.ORG_UNIT_CODE
AND SGD.V_PROD_CODE = DPB.PRODUCT_CODE
AND SGD.V_FINANCIAL_ELEMENT_CODE = DFEB.FINANCIAL_ELEM_CODE
AND SGD.V_COMMON_COA_CODE = DCCB.COMMON_COA_CODE
AND SGD.V_CCY_CODE = FC.ISO_CURRENCY_CD
AND SGD.V_SCENARIO_CODE = FCC.CONSOLIDATION_DISPLAY_CODE;
```

> **Important:** As seen in the code above, the view references the "_CODE" columns on the dimension tables. For example, COMMON_COA_CODE on DIM_COMMON_COA_B and ORG_UNIT_CODE on DIM_ORG_UNIT_B. These code columns must be populated for data to exist in STG_GL_DATA_V.

The "Update_Dimension_Code" (fn_updatedimensioncode) program populates these Code columns using data from values in the "Code" dimension Attribute (ex. COMMON COA CODE, ORG UNIT CODE, etc.)

The BALTYPECD column has a default value of 0 in the View, as this column is not null in LEDGER_STAT. Baltypecd is not a Dimension. It indicates the credit or debit of the same account details. Since same account can hold both credit and debit, this column should be populated in the source with a value. It is the part of the unique Index and Not Null column in LEDGER_STAT.

Step 3: If using the Type 3 Ledger Stat Load for the first time, run the GTT table creation procedure.

The GTT table creation procedure creates the Global Temporary Table LS_LOAD_TABLE_GTT_V.

The fn_ledger_load_create_gtt function creates the table LS_LOAD_TABLE_GTT_V and the index UK_GTT for use in the Type 3 Ledger Stat Load.

Note: If the GTT table has not been created and you try to execute the Ledger Stat Load, you will get the following error in FSI_MESSAGE_LOG:

```
WRAPPER_LEDGER_STAT_LOAD- Error: -942: ORA-00942: table or
view does not exist
```

Step 4: Populate FSI_LS_LOAD_BATCH

You need to populate the following columns:

RUN_FLAG	Y
SEQUENCE	Sequence value (ex. 1)
LOAD_TABLE_NAME	STG_GL_DATA
ONE_MONTH_ONLY	N
UPDATE_MODE	ADD or REPLACE
INSERT_ONLY	Y or N
CREATE_OFFSETS	N
IS_CALENDAR_MONTH	Y
START_CALENDAR_MONTH	Starting date to load in format YYYYMMDD.
END_CALENDAR_MONTH	Ending date to load in format YYYYMMDD.

Step 5: Run the Ledger Stat Load

Use the following command to run the Type 3 Ledger Stat Load in SQL*Plus as the atomic user:

```
DECLARE
x NUMBER :=0;
BEGIN
x :=
ofsa_util.wrapper_ledger_stat_load('BATCH_ID ','MIS_DATE','TABLE_NAME',
TABLE_TYPE', 'UPDATE_MODE', 'INSERT_ONLY', 'START_DATE', 'END_DATE')
dbms_output.put_line ('The return variable is ' || x);
END;
```

Example

DECLARE x NUMBER :=0; BEGIN x :=
ofsa_util.wrapper_ledger_stat_load('ARALSLOADTYPE3_4','20110111','STG_GL_DATA',

'CALENDAR_MONTHS', 'ADD', 'Y', '20101231', '20101231'); dbms_output.put_line ('The return variable is ' || x); END;

After the Ledger Load completes, check the tables FSI_MESSAGE_LOG and FSI_LS_LOAD_BATCH for errors.

Creating View on LEDGER_STAT table

A view is created on the LEDGER_STAT table called LSL. The purpose of this view is to provide shorter column names for the load procedure. The LSL view must contain the same columns as LEDGER_STAT. Column alias for each columns within the view should match the COLUMN_ALIAS user-defined property that is set for each column of LEDGER_STAT table in the ERwin model.

For any user-defined dimensions in your LEDGER_STAT you must complete the following steps.

- In ERwin model, look up the COLUMN_ALIAS User Defined Property (UDP) for added dimension columns within LEDGER_STAT table.
- Specify the value of the property COLUMN_ALIAS.
- Modify the view to include new dimension columns. Use the same COLUMN_ALIAS that was mentioned in the ERwin model in the load table view.

Creating Load Table

This step is applicable for loading ledger data from Type I or Type II load table. Staging table STG_GL_DATA (used for Type III load) is packaged with the application. Multiple load tables (Type I or Type II) can be created as required by the System Administrator. Table structure for the Type I and Type II load tables is given in the following sections:

```
-- ---------------------------------------------------------------------
-- Uncomment the m1..m12 columns if you plan to load a range of months
(Type II Load Table).
-- Add lines for all of the LEDGER_STAT user-defined leaf columns in the
place
-- indicated below. Don't forget to add commas if you need to.
-- ---------------------------------------------------------------------
CREATE TABLE &load_table_name(
      ds              VARCHAR2(12)   NOT NULL,   -- data_source
      year_s          NUMBER(5)      NOT NULL,
      accum_type      char(1)        NOT NULL,
      consolidat      NUMBER(5)      NOT NULL,
      isocrncycd      VARCHAR2(3)    DEFAULT '002' NOT NULL,
      financ_id       NUMBER(14)     NOT NULL,
      org_id          NUMBER(14)     NOT NULL,
      gl_acct_id      NUMBER(14)     NOT NULL,
      cmn_coa_id      NUMBER(14)     NOT NULL,
      prdct_id        NUMBER(14)     NOT NULL,
      baltypecd       NUMBER(5)      DEFAULT 0 NOT NULL,
   --
   -- m1              NUMBER(15,4),
   -- m2              NUMBER(15,4),
   -- m3              NUMBER(15,4),
   -- m4              NUMBER(15,4),
   -- m5              NUMBER(15,4),
   -- m6              NUMBER(15,4),
   -- m7              NUMBER(15,4),
   -- m8              NUMBER(15,4),
   -- m9              NUMBER(15,4),
   -- m10             NUMBER(15,4),
   -- m11             NUMBER(15,4),
   -- m12             NUMBER(15,4),
   --
      one_month_amt   NUMBER(15,4)
   --
    --
-------------------------------------------------------------------------
   --  Other leaf columns (PROPERTY_COLUMN from REV_COLUMN_PROPERTIES
for LEDGER_STAT):
    --
-------------------------------------------------------------------------
   -- . . .
   --
   )
```

Creating Unique Index on Load Table

This step is applicable for loading ledger data from Type I or Type II load table. A unique index has to be created on each load table specifying the column alias for each column within the load table. Column alias should match the column alias specified for columns within LEDGER_STAT table. LEDGER_STAT load procedure identifies the source columns that need to be loaded using the column aliases and not by the physical column names. Column alias for LEDGER_STAT columns are specified in the user-defined property (UDP) COLUMN_ALIAS within ERwin model. Refer to ERwin model for getting the column alias for each of the LEDGER_STAT columns. Definition of the unique index is given below:

```
CREATE UNIQUE INDEX &load_table_name
   ON &load_table_name ( ds,
                year_s,
                accum_type,
                consolidat,
                isocrncycd,
                financ_id,
                org_id,
                gl_acct_id,
                cmn_coa_id,
                prdct_id
                baltypecd,
            -- ------------------------------------------------------
            -- Include all additional LEDGER_STAT primary key
            -- leaf columns here (use PROPERTY_COLUMN from
REV_COLUMN_PROPERTIES):
            -- ------------------------------------------------------
            --  . . .
            --
     )
```

The unique key of the load table must be identical to the unique key of LEDGER_STAT, with the exception that instead of IDENTITY_CODE, which is in LEDGER_STAT, the load table has a column called DS (Data Source).

Creating Views on Load Table

This step is applicable for loading ledger data from Type I or Type II load table. In addition to load tables, views have to be created on the staging tables similar to the view LSL that was created on LEDGER_STAT. A view has to be created on each load table specifying the columns alias for each column within the load table. Column alias should match the column alias specified for columns within LEDGER_STAT table. LEDGER_STAT load procedure identifies the source columns that need to be loaded using the column alias. Column alias for LEDGER_STAT columns are specified in the user-defined property (UDP) COLUMN_ALIAS within ERwin model. Refer to ERwin model for getting the column alias for each of the LEDGER_STAT columns. View definition is given below:

```
-- ------------------------------------------------------------------
-- Uncomment the m1..m12 columns if you plan to load a range of months
(Type II Load table).
-- Add lines for all of the LEDGER_STAT user-defined leaf columns in the
place
-- indicated below. Don't forget to add commas if you need to.
-- ------------------------------------------------------------------
CREATE OR REPLACE VIEW &load_table_name._v AS
   SELECT ds,
          year_s,
          accum_type,
          consolidat,
          isocrncycd,
          financ_id,
          org_id,
          gl_acct_id,
          cmn_coa_id,
          prdct_id,
          baltypecd,
       --
       -- NVL(m1,0) AS m1,
       -- NVL(m2,0) AS m2,
       -- NVL(m3,0) AS m3,
       -- NVL(m4,0) AS m4,
       -- NVL(m5,0) AS m5,
       -- NVL(m6,0) AS m6,
       -- NVL(m7,0) AS m7,
       -- NVL(m8,0) AS m8,
       -- NVL(m9,0) AS m9,
       -- NVL(m10,0) AS m10,
       -- NVL(m11,0) AS m11,
       -- NVL(m12,0) AS m12,
       --
          NVL(one_month_amt,0) AS one
       --
       --
-------------------------------------------------------------------------
       -- Other leaf columns (PROPERTY_COLUMN from REV_COLUMN_PROPERTIES
for LEDGER_STAT):
      --
-------------------------------------------------------------------------
       -- . . .
       --
     FROM &load_table_name
     WHERE NVL(one_month_amt,0) <> 0;
       --
       -- OR NVL(m1,0) <> 0
       -- OR NVL(m2,0) <> 0
       -- OR NVL(m3,0) <> 0
       -- OR NVL(m4,0) <> 0
       -- OR NVL(m5,0) <> 0
       -- OR NVL(m6,0) <> 0
       -- OR NVL(m7,0) <> 0
       -- OR NVL(m8,0) <> 0
       -- OR NVL(m9,0) <> 0
       -- OR NVL(m10,0) <> 0
       -- OR NVL(m11,0) <> 0
       -- OR NVL(m12,0) <> 0;
```

In case, the custom dimensions are added to the load table, views need to be modified to reflect the same.

Setting up Global Temporary Table

This step is applicable for loading ledger data from Type III. Calendar dates present in the data of Load table are converted to the corresponding Fiscal Year/Month. Conversion from calendar date to fiscal year & month is done based on the START_MONTH column present in FSI_FISCAL_YEAR_INFO table. These derived fiscal year & fiscal month are then inserted in an intermediate Global Temporary Table (GTT) after aggregating the rows of same months/years. Therefore, if 12 rows are present for the same fiscal year each corresponding to a different month, then global temporary table may have maximum of one row corresponding to the fiscal months, these 12 rows represent.

GTT needs to contain valid dimension member identifiers and numeric codes. Since staging table contains alphanumeric identifiers and codes, a view is created on STG_GL_DATA table joining with other relevant dimension and CD/MLS tables before being used in the GTT creation.

Global temporary table can be created in 2 ways as described below:

1. Using PL/SQL

```
Declare
output number;
Begin
 Output:= fn_ledger_load_create_gtt('BATCH_ID', 'AS_OF_DATE',
'TABLE_NAME');
End;
```

AS_OF_DATE is the date for which GTT is created, in YYYYMMDD format.

TABLE_NAME is the staging table name STG_GL_DATA.

An example of running the function from SQL*Plus is as follows:

```
SQL> var output number;
SQL> execute :output:= fn_ledger_load_create_gtt('BATCH_ID',
'20100519', 'STG_GL_DATA');
```

2. Using OFSAAI ICC Framework

To execute the procedure from OFSAAI ICC framework, run the batch mentioned below and specify the following parameters:

- Datastore Type:- Select appropriate datastore from list
- Datastore Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- fn_ledgerLoadGTTCreation

- Parameter List:- AS_OF_DATE and TABLE_NAME

TABLE_NAME is the staging table name STG_GL_DATA.

AS_OF_DATE should be passed as 'YYYYMMDD' format.

> **Note:** BATCHID will be passed explicitly in ICC framework. The appropriate table parameters are enclosed in single quotes.

Tables Related to LEDGER_STAT Load Procedure

LEDGER_STAT Loader utility uses the following tables:

- FSI_FISCAL_YEAR_INFO – The table contains the fiscal year information. This is a setup table.
- FSI_LS_LOAD_BATCH – The table contains the parameters for the load batch that needs to be executed for loading ledger data from staging or load table into LEDGER_STAT. This is a setup table.
- STG_GL_DATA – The staging table contains the ledger data for various as-of-dates.
- LEDGER_STAT – The processing table contains the ledger data for various fiscal months. This is loaded from staging table.

Populating Stage Tables

Data for ledger can come from external systems. Such data has to be in the format of the staging table. This data can be loaded into staging through F2T component of OFSAAI framework. Users can view the loaded data by querying the staging tables and various log files associated with F2T component.

Executing LEDGER_STAT Load Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The procedure/batch requires the following 8 parameters:

1. *BATCH_ID-Any unique number to identify the execution run.*
2. *MIS_DATE- Date on which the loading is done expressed in YYYYMMDD format.*
3. *TABLE_NAME- STG_GL_DATA(Type III) or any other load table (TYPE I or TYPE II)*
4. *TABLE_TYPE- FISCAL_ONE_MONTH or FISCAL_RANGE (TYPE I or TYPE II)*

CALENDAR_MONTHS (TYPE III)

5. *UPDATE_MODE-ADD/REPLACE*
6. *INSERT_ONLY- Y/N*
7. *START_DATE- Calendar start date in YYYYMMDD*
8. *END_DATE- Calendar end date in YYYYMMDD*

The input parameter logic for the Type III, Type II and Type I tables.

CALENDAR_MONTHS

- If Start_Date and End_Date are null then month part of MIS_Date is taken for processing a particular month. (Ex: if MIS_DATE is 20101231 then the December calendar month data is processed).
- In this case the Start_Date and End_Date becomes optional.

FISCAL_ONE_MONTH

- The Start_Date and End_Date parameters will hold numeric values identifying the fiscal month. The value of these parameters will be between 1 and 12 (i.e. M1 till M12).
- The Start_Date and End_Date should be same.
- In this case the Start_Date and End_Date are mandatory.

FISCAL_RANGE

- The Start_Date and End_Date parameters will hold numeric values identifying the fiscal month. The value of these parameters will be between 1 and 12 (i.e. M1 till M12).
- The Start_Date and End_Date parameters will specify the range of fiscal months which are to be processed. Ex: M1 till M6 in case the Start_Date and End_Date values are 1 and 6.
- In this case the Start_Date and End_Date are mandatory.

Ledger Load can be executed in 2 different ways:

1. Using PL/SQL:

```
By using the function
ofsa_util.wrapper_ledger_stat_load('BATCH_ID','MIS_DATE',
TABLE_NAME', TABLE_TYPE', 'UPDATE_MODE',
'INSERT_ONLY','START_DATE','END_DATE');
```

Example:

```
DECLARE
x NUMBER :=0;
BEGIN
x :=
ofsa_util.wrapper_ledger_stat_load('batch_id_1','20090202','STG_GL_D
ATA','CALENDAR_MONTHS','ADD','Y','20070430','20080331');
dbms_output.put_line ('The return variable is ' || x);
END;
```

2. To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:
   - Datastore Type:- Select appropriate datastore from list
   - Datastore Name:- Select appropriate name from the list
   - IP address:- Select the IP address from the list
   - Rule Name:- **fn_ledgerDataLoader**
   - Parameter List:- <Same as mentioned above in the parameter list>

Executing LEDGER_STAT Load Procedure for MULTI CURRENCIES

The data for the Ledger can have more than one currency at a time that is multi currencies input.

To execute multi currencies, one needs to enable the below flag using the below statement:

update fsi_db_info f1 set f1.multi_currency_enabled_flg =1;

commit;

Exception Messages

The ledger load program throws both user defined exceptions and Oracle database related exceptions. These exception messages could be seen in FSI_MESSAGES_LOG table with the help of the batch_id which was used during execution. The exception list includes all possible validations on the parameters that were passed and database related exceptions.

Tables Cleanup After Truncation Of Ledger_Stat

The LEDGER_STAT procedure makes entries into certain audit tables. Whenever the

user truncates/deletes the Data from LEDGER_STAT, he needs to additionally remove the auditing entries from the tables FSI_DATA_IDENTITY, FSI_M_SRC_DRIVER_QUERY and FSI_LS_MIGRATION_RESULTS. This procedure enables the user to clean up these audit tables.

Executing the clean up of Ledger_Stat Load Procedure

```
To run the procedure from SQL*Plus, login to SQL*Plus as the Schema
Owner:
Fn_ledger_stat_cleanup(batch_run_id VARCHAR2,
as_of_date   VARCHAR2)
SQLPLUS > declare
result number;
begin
result:=Fn_ledger_stat_cleanup ('LEDGER_CLEANUP_BATCH1','20121212');
end;
/
 BATCH_RUN_ID is any string to identify the executed batch.
 AS_OF_DATE in the format YYYYMMDD.
```

Ledger Stat Clean up Procedure

To execute Ledger Stat Clean up from OFSAAI ICC framework, a seeded Batch is provided.

The batch parameters as mentioned below:

- **Datastore Type:** Select the appropriate datastore from list
- **Datastore Name:** Select the appropriate name from the list
- **IP address:** Select the IP address from the list
- **Rule Name:** Fn_ledger_stat_cleanup

Cash Flow Loader

Customers typically source some of their cash flow result data from 3rd party information providers or from internal models that are built for the purpose. The cash flow loader provides a way to load these cash flows directly into the ALM Cash Flow Detail tables. Cash flows will be loaded into the Staging Cash flow table by the end user. The implementation team will typically be responsible for the first stage mapping from source to our staging table. This staging cash flow table will store aggregated cash flows at product/Org unit/Currency level and Instrument cash flows also.

The following topics are covered in this section:

- Exception Messages, page 4-83

Tables related to Cash Flow Loader

- **STG_ACCOUNT_CASH_FLOW:** This table will be used to store the cash flow generated by the different sources for loading purpose. Customer should feed the cash flow data.
- **Output tables:** Aggregated Cash Flows will be populated in the below output tables
  - RES_DTL_XX
  - CONS_DTL_XX
  - FSI_O_RESULT_MASTER
  - FSI_O_CONSOLIDATED_MASTER

  XX denotes the process id.
- **SETUP_MASTER:** This table will be used in the case of instrument cash flows. For Instrument cash flows, an entry against V_COMPONENT_VALUE of the SETUP_MASTER table should have values either 0 or 1 which indicate if id numbers or account numbers are provided, respectively.
- **FSI_ALM_DETERMINISTIC_PROCESS:** This table will be used for loading cash flows in CONSOLIDATED tables. To populate consolidated tables, the CONSOLIDATED_OUTPUT_FLG should be 1 in the fsi_alm_deterministic_process table against the cash flow process ids.
- **FSI_CASH_FLOW_LOADER_SETUP:** This table will have all the process ids for cash flow loader. Only those processes will be executed which have status 'N' in FSI_CASH_FLOW_LOADER_SETUP table.
- **FSI_M_USER_ACTIVE_TIME_BUCKETS:** For cash flow loader, user should be mapped to an active time bucket in the FSI_M_USER_ACTIVE_TIME_BUCKETS table.
- **TIME BUCKETS:** The following tables will store the time bucket details:
  - FSI_TIME_BUCKET_MASTER
  - FSI_M_LR_IRR_BUCKETS
  - FSI_LR_IRR_BUCKETS_AUX

- FSI_TIME_BKT_ISB
- FSI_TIME_BKT_LR_LRR_DATES

Data Validation Steps

- Check if Batch run id or mis date or User name is null. If Yes, then write message in fsi_message log and exit.
- Check if the given user name exists in fsi_m_user_active_time_buckets table. If user does not exist, then write error message in Fsi_Message table and exit.
- The time bucket mapped to user in fsi_m_user_active_time_buckets table should be present in fsi_income_simulation_buckets table. If time bucket is not present in fsi_income_simulation_buckets table, then write error message in fsi_message table and exit.
- Only financial elements corresponding to Income Simulation Buckets (ISB), Liquidity risk (LR) and Interest Rate Risk (IRR) will be processed.
- If process id is given as parameter, then run the loader program for the given process id. If process id is not given, then the loader program will be run for all the process ids mapped in the set up table with status 'N'.
- Verify that the given process is present in fsi_alm_deterministic_process table and fsi_m_alm_process table. If not present, then write error message in fsi_message table and exit.
- Check if the given process id mapped in fsi_cash_flow_loader_setup table is of status 'N'.
- Check if the given cash flow date for the process and scenario given is present in the fsi_time_bkt_isb and fsi_time_bkt_lr_irr_dates tables for the respective time bucket.
- Check if the set up table is mapped correctly or not.

  Example
  - Check if the given process id mapped to a scenario is present in the stg_account_cash_flow table.
  - Check if the entire scenario mapped in the set up table with a functional currency value has its base currency present.
- Check if 'consolidated_output_flag' in the fsi_alm_deterministic_process table is 1. If yes, write to consolidated tables.

- Verify that all the dimension ids given in the stg_account_cash_flow table is valid and present in the respective dimension tables.
- In the case of instrument level cash flows, the following conditions should be satisfied to proceed:
  - In the setup_master table for the v_component_code =123, the v_component_value should be either 0 or 1 which indicates account number or identity number is given in the stg_account_cash_flow table, respectively.
  - Identity code and id_number/account number should be present in the stg_account_cash_flow table.

Executing Cash Flow Loader

The user can execute this Cashflow Loader from either SQL*Plus or from within a PL/SQL block or from the ICC Batch screen within OFSAAI framework.

Method 1

Cash Flow Loader can be executed directly from SQL Plus. User must login to the database using schema user id and password. The procedure requires 4 parameters:

- As of Date (mis_date)
- User Name – Should be present in fsi_m_user_active_time_buckets table
- Process id
- Batch Execution Identifier (batch_run_id)

```
declare
result number :=0;
begin
   result := fn_cash_flow_loader(batch_run_id => :batch_run_id,
                                 mis_date => :mis_date,
                                 p_user_name => :p_user_name,
                                 p_process_sys_id => :p_process_sys_id);
    if result = 0 then
    dbms_output.put_line('Cash Flow Loader Failed');
    else
    dbms_output.put_line('Cash Flow Loader Succesfully completed');
    end;
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- mis_date in the format YYYYMMDD.
- P_USER_NAME – The user name present in

FSI_M_USER_ACTIVE_TIME_BUCKETS table.

- P_PROCESS_SYS_ID can be null or can have value to process specific process id.

Case 1. When Process id is null:

```
declare
result number :=0;
begin
  result := fn_cash_flow_loader('INFODOM_ CASH_FLOW_LOADER',
                                                          '20100419',
                                                        'ALMUSER'
                                                        NULL);

if result = 0 then
dbms_output.put_line('Cash Flow Loader Failed');
else
dbms_output.put_line('Cash Flow Loader Succesfully completed');
end;
```

Case 2. When Process id is not null:

```
declare
result number :=0;
begin
  result := fn_cash_flow_loader('INFODOM_ CASH_FLOW_LOADER',
                                                          '20100419',
                                                        'ALMUSER'
                                                        '120003);

if result = 0 then
dbms_output.put_line('Cash Flow Loader Failed');
else
dbms_output.put_line('Cash Flow Loader Succesfully completed');
end;
```

SETUP_MASTER Sample Data

V_COMPONENT_CODE	V_COMPONENT_DESC	V_COMPONENT_VALUE
123	Cash Flow Loader	1

For instrument cash flows, the V_COMPONENT_VALUE should be either 1 or 0. If the value is '1' then Identity code and Account number should be populated in the stg_account_cash_flows table . If the Value is '0' then Identity code and ID Number should be populated in the stg_account_cash_flows table.

FSI_CASH_FLOW_LOADER_SETUP Sample Data

RUN_DATE	PROCESS_ID	SCENARIO_NUM	STATUS
12/10/2001	120003	1	N
12/10/2001	120123	1	N

Note: Only the process id with Status 'N' will process while executing the loader program. After the successful execution of the loader program the value in the STATUS columns will be changed to 'Y' in FSI_CASH_FLOW_LOADER_SETUP for the process id.

FSI_M_ALM_PROCESS Sample Data

TM_PROCESS_SYS_ID	ALM_PROCESS_TYPE_CD	FILTER_SYS_ID	FILTER_TYPE	PREPAY_SYS_ID	PROD_CHAR_ID	REPORTING_CURRENCY_(
120003	1	0	0	0	0	USD
120123	1	0	0	0	0	USD

Continuation...

REPORTING_CURRENCY_CD	STATUS	TRANSACTION_SYS_ID	ALT_SOURCE_FLG	BI_TRANSFORM_STATUS
USD	0	0	0	
USD	0	0	0	

STG_ACCOUNT_CASH_FLOWS Sample Data

Instrument Cashflow Data

V_ACCOUNT_NUMBER	FIC_MIS_DATE	N_ACCT_	N_ACCOU	N_SCENARIO_NO	N_CURRENCY_	V_FINANCIAL_ELE	V_CCY	N_CASH_FLOW_SEQL	D_CASH_FLOW_DA	V_DATA_ORIGI	V_CASH_FLOW_TYPE
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	34	10/25/1997	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	35	11/25/1997	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	36	12/25/1997	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	37	1/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	38	2/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	39	3/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	40	4/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	41	5/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	42	6/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRUNOFF	USD	43	7/25/1998	SRC	

Continuation...

V_CASH_FLOW_TYPE	N_CASH_FLOW_AMOUNT	V_ORG_UNIT_CODE	V_PROD_CODE	N_INSTRUMENT_TYPE_CD	V_GL_ACCOUNT_CODE	V_COMMON_COA_CO
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		
	11000.350	ORG1	PROD1	120		

Aggregate Data Sample

V_ACCOUNT_NUMBER	FIC_MIS_DATE	N_ACCT_	N_ACCOU	N_SCENARIO_NO	N_CURRENCY_	V_FINANCIAL_ELE	V_CCY	N_CASH_FLOW_SEQL	D_CASH_FLOW_DA	V_DATA_ORIGI	V_CASH_FLOW_TYPE	N_CASH_
0	12/31/2001	0	0	3	1	GAPRUNOFF	USD	1	1/31/2001	SRC2		
0	12/31/2002	0	0	3	1	GAPRUNOFF	USD	2	4/30/2001	SRC2		
0	12/31/2003	0	0	3	1	GAPRUNOFF	USD	3	7/31/2001	SRC2		
0	12/31/2004	0	0	3	1	GAPRUNOFF	USD	4	10/31/2001	SRC2		
0	12/31/2005	0	0	3	1	GAPRUNOFF	USD	5	1/31/2002	SRC2		
0	12/31/2006	0	0	3	1	GAPRUNOFF	USD	6	4/30/2002	SRC2		
0	12/31/2007	0	0	3	2	GAPRUNOFF	USD	7	7/31/2002	SRC2		

Continuation...

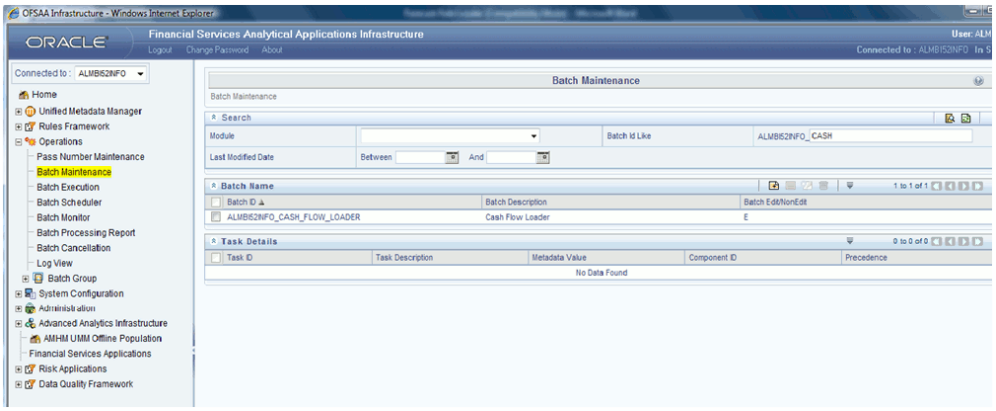
SH_FLOW_AMOUNT	V_ORG_UNIT_CODE	V_PROD_CODE	N_INSTRUMENT_TYPE_CD	V_GL_ACCOUNT_CODE	V_COMMON_COA_CO
100834348.3	ORG2	PROD2			
120833421.3	ORG2	PROD2			
91038411.3	ORG2	PROD2			
85128473.5	ORG2	PROD2			
13048274.34	ORG2	PROD2			
13700688.06	ORG2	PROD2			
13974701.82	ORG2	PROD2			

Method 2

To execute the Cash Flow Loader from OFSAAI ICC framework, a seeded Batch is provided.

Execution Steps

1. Select "<INFODOM>_CASH_FLOW_LOADER" as the Batch ID and "Cash Flow Loader" is the description of the batch.

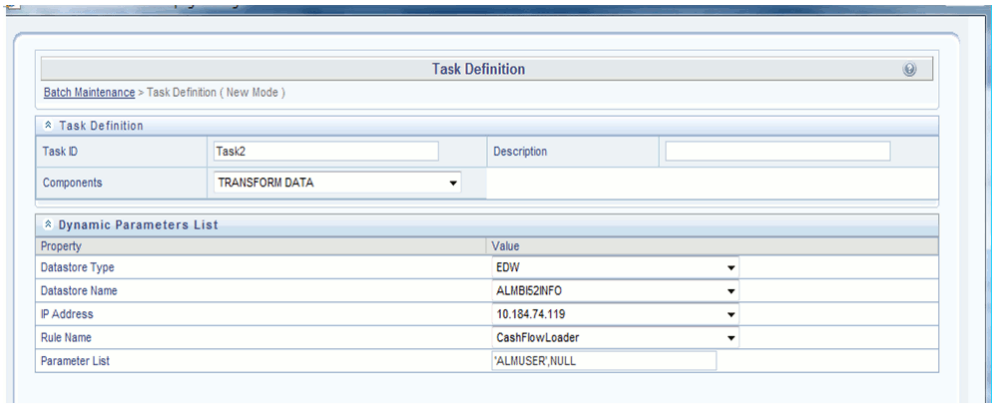


2. The batch has a single task. Edit the task.

3. If the user wants to load all the process ids given in the FSI_CASH_FLOW_LOADER_SETUP table for the given as of date, then Process_id parameter should be null.

4. Specify the following parameters:

   - Data store Type:- Select appropriate data store from list
   - Data store Name:- Select appropriate name from the list
   - IP address:- Select the IP address from the list
   - Rule Name:- *CashFlowLoader*

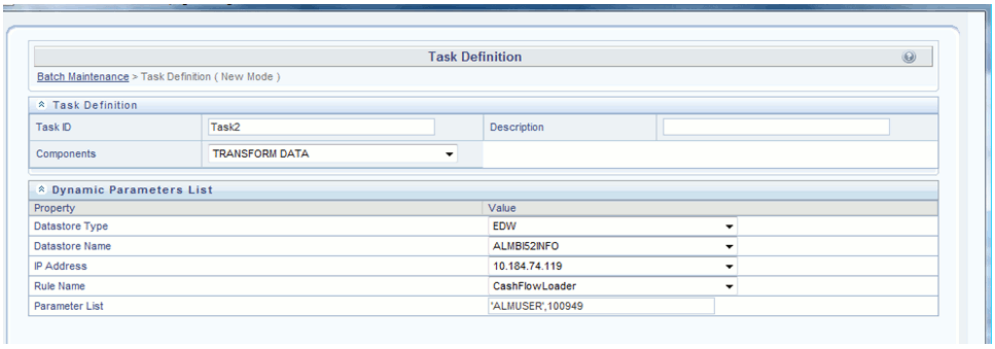


5. If the user wants to process the specific process id mentioned in the FSI_CASH_FLOW_LOADER_SETUP table for the given as of date, then the process_id parameter should be given.

6. Specify the following parameters:

   - Data store Type:- Select appropriate data store from list

- Data store Name:- Select appropriate name from the list
- IP address:- Select the IP address from the list
- Rule Name:- *CashFlowLoader*



7. Save the batch
8. Execute the Batch defined for the required As of Date.

Exception Messages

All the exceptions will be logged in Fsi_Message table. Cash Flow Loader program can raise the following exceptions:-

- **Exception 1 - Load fails:Cannot pass null for the parameter As of Date, User name, Batch run id**

  As of date, User Name and Batch run id cannot be passed as null.

- **Exception 2 :- Load Fails: User name does not exist in the Table fsi_m_user_active_time_buckets**

  User name in the Parameter should be mapped with an active time bucket in the fsi_m_user_active_time_buckets table.

- **Exception 3 :- Load Fails: Data for the selected As of date does not exist in stg_account_cash_flows Table**

  Stage Account Cash Flow does not have data for the given As Of Date.

- **Exception 4:- Load Fails: Time bucket sys id is not present in fsi_income_simulation_buckets**

  Active time bucket mapped to the user name should be present in the Fsi_Income_Simulation_Bucket table.

- **Exception 5:- Load Fails: dates calculation failed**

  Date Calculation for Fsi_Time_Bkt_Isb and Fsi_Time_Bkt_Lr_Irr_Dates failed for the given time bucket sys id.

- **Exception 6:- Load Fails: The process id process_sys_id does not exist in the Table fsi_m_alm_process**

  Process id which is either passed as parameter or picked up from the Fsi_Cash_Flow_loader_Setup table should be present in the fsi_m_alm_process table.

- **Exception 7:- Load Fails. The process id process_sys_id does not exist in the Table Fsi_alm_deterministic_process**

  Process id which is either passed as parameter or picked up from the Fsi_Cash_Flow_loader_Setup should be present in the Fsi_alm_deterministic_process table.

- **Exception 8:- Load Fails. The process id process_sys_id does not exists in the table fsi_cash_flow_loader_setup or the status is not set to "N" for the process**

  Process id which is either passed as parameter, either does not exist in the Fsi_Cash_Flow_loader_Setup table or the status is not 'N'.

- **Exception 9:- Load Fail: The Cash flow dates: cash_flow_missing for the process id process_sys_id is not defined in the FSI_TIME_BKT_ISB_DATES /FSI_TIME_BKT_LR_IRR_DATES table**

  Cash Flow date present in the Stg_Account_Cash_Flow is not matching with the dates presentin the Fsi_Time_Bkt_Isb/ Fsi_Time_Bkt_Lr_Irr_Dates tables for the active time bucket id mapped to the user name.

- **Exception 10:- Load Fail: The Data for the N_SCENARIO_NO mapped to the process process_sys_id does not exist in the STG_ACCOUNT_CASH_FLOWS table**

  Scenario Number mismatch for the Fsi_Cash_Flow_Loader and Stg_Account_Cash_Flow

- **Exception 11:- Load Fail: Does not have base currency**

  Base currency should be present in Stg_Account_Cash_Flow.

- **Exception 12:- CONSOLIDATED_OUTPUT_FLG in fsi_alm_deterministic_process able is "1" but no data for n_currency_type_cd in STG_ACCOUNT_CASH_FLOWS table**

  Consolidated Flag for the process id is set to 1 but n_currency_type_cd in Stg_Account_Cash_Flow is not set.

- **Exception 13:- Load Fail: For the N_SCENARIO_NO mapped in the set up table, the dimension code given is incorrect**

  Dimensions present in Stg_Account_Cash_Flow are not present in the corresponding dimension tables.

- **Exception 14:- All the account numbers are not present in the STG_ACCOUNT_CASH_FLOWS table**

  Records in Stg_Account_Cash_Flow do not have account number populated.

- **Exception 15:- All the Identity codes are not present in the STG_ACCOUNT_CASH_FLOWS table**

  Records in Stg_Account_Cash_Flow do not have identity code populated.

- **Exception 16:- All the Id Numbers are not present in the STG_ACCOUNT_CASH_FLOWS table**

  Records in Stg_Account_Cash_Flow do not have identity number populated.

- **Exception 17 :- Instrument type code given for the process l_process_sys_id is wrong**

  Instrument code present in the Stage table is not mapped in FSI_INSTRUMENT_TYPE_MLS

- **Exception 18: No Data in the instrument table for the given FIC MIS DATE. Loading data to FSI_O_RESULT_MASTER failed**

  Instrument table corresponding to instrument code in Stg_Account_Cash_Flow does not have data for the given As of date.

Pricing Management Transfer Rate Population Procedure

This function populates FSI_M_PROD_TRANSFER_RATE table from FSI_PM_GENERATED_INSTRMTS table for particular Effective date.

After executing this procedure, you should query FSI_M_PROD_TRANSFER_RATE table.

Executing the POPULATE_PM_TRANS_RATE_TABLE (earlier known as POPULATE_TPOL_TRANS_RATE) Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from ICC Batch screen within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner.

The procedure requires the following 6 parameters:

1. Batch Id (Batch_Id) – can be used to see the log of the procedure executed.
2. Misdate (Mis_date) - the date for which batch is run.
3. Run Id (p_v_run_id) - Unique Run ID for the run.
4. Process Id (p_v_process_id) - Unique Process ID for the batch.
5. Run Execution Id (p_v_run_execution_id) - Unique Run Execution Id for the Run.
6. Run skey (p_n_run_skey) – Unique run skey generated by the run.

The syntax for calling the procedure is:

```
Declare
output number;
Begin
Output:= POPULATE_PM_TRANS_RATE_TABLE (Batch_Id varchar2,
                  Mis_date varchar2,
                                        p_v_run_id varchar2,
                                        p_v_process_id varchar2,
                                         p_v_run_execution_id varchar2,
                                        p_n_run_skey varchar2);
End;
```

Mis_date should be passed as 'YYYYMMDD' format.

An example of running the function from SQL*Plus is as follows:

```
SQL> var output number;
SQL> execute: output:= POPULATE_PM_TRANS_RATE_TABLE('Batch_Id',
'20100131,' $RUNID=1306182237482', '$PHID=1228363751510',
'$EXEID=RQEXE016','$RUNSK=99');
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows.

```
SQL> declare
output number;
begin
Output:= POPULATE_PM_TRANS_RATE_TABLE ('Batch_Id','Mis_date',
'p_v_run_id','p_v_process_id','p_v_run_execution_id',' p_n_run_skey');
End;
/
```

To execute the procedure from OFSAAI ICC framework, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- Datastore Type :- Select appropriate datastore from the list
- Datastore Name :- Select appropriate name from the list
- IP address :- Select the IP address from the list
- Rule Name :- POPULATE_PM_TRANS_RATE_TABLE

Note: BATCHID and MISDATE will be passed explicitly in ICC framework

ALMBI Transformation

ALM_BI_TRANSFORMATION data definition transforms the Asset Liability Management (ALM) processing results of an executed ALM process to ALMBI fact tables.

This internally calls PL/SQL function FN_ALM_BI_TRANSFORMATION.

```
function FN_ALM_BI_TRANSFORMATION(p_batch_run_id varchar2,
                                  p_as_of_date   varchar2,
                                  PID            number,
                                  p_re_run_flag  char)
```

Where the parameters are,

1. p_batch_run_id - It is the batch run id. Batch Run ID value is passed from the Batch execution UI. Therefore, it is not required to define it as a parameter value in Batch Maintenance.

2. p_as_of_date - This parameter value is passed from the Batch execution UI. Therefore, it is not required to define it as a parameter value in Batch Maintenance.

3. PID - Pass the ALM Process Sys ID for which the transformation has to be done.

4. p_re_run_flag - This parameter value determines whether the transformation for the ALM process is for the first time or not.

   Possible values are 'Y' or 'N'

   Where

   'Y' - Yes (This means that the transformation was already done and the user is trying to redo the transformation once again for the ALM process).

   'N' - No (This means that the user is executing the transformation for the first time for the ALM process).

   Note: The values for parameters **PID** and **p_re_run_flag** has to be entered in the Parameter List during the batch definition.

Example

If the user is trying to do transformation of ALM process 200009 for the first time, then the values that must be entered in the Parameter List are 200009, 'N'.

If the user is trying to do transformation of ALM process 200011, for which he had already done the transformation, then the values that must be entered in the Parameter

List are 200011, 'Y'.

Hierarchy Transformation

Hierarchy Flattening Transformation is used to move the hierarchy data from the parent child storage structure in EPM AMHM (Attribute, Member and Hierarchy Management) model to a level based storage structure in OFS Profitability Analytics. In EPM AMHM model, hierarchy data for any hierarchy created on seeded or user defined dimensions using the AMHM is stored within hierarchy tables of respective dimensions. This is moved to the REV_HIER_FLATTENED table in OFS Profitability Analytics after flattening by the Hierarchy flattening process.

batch_hierTransformation is a seeded Data Transformation program installed as part of the OFSPA solution installer.

Executing the Hierarchy Flattening Transformation

You can execute this procedure from SQL Plus/PLSQL/ICC Batch screen within OFSAAI framework.

1. Using SQL Plus/PLSQL

   Function Name: rev_batchHierFlatten

   Parameters: batch_run_id, mis_date, pDimensionId, pHierarchyId

```
function rev_batchHierFlatten(batch_run_id varchar2,
                              mis_date varchar2,
                              pDimensionId varchar2,
                              pHierarchyId varchar2,
                              )
```

   Where the parameters are,

   - batch_run_id - It is the batch run id. Batch Run ID value is passed from the Batch execution UI. Therefore, it is not required to define it as a parameter value in Batch Maintenance.

   - mis_date - This parameter value is passed from the Batch execution UI. Therefore, it is not required to define it as a parameter value in Batch Maintenance. Follow the date format, YYYYMMDD

   - pDimensionId- Enter the Dimension id . To find dimension id, execute the following query in database to find the value and use the value in dimension id column for the dimension name / description to be processed:

     Select b.dimension_id,t.dimension_name,t.description from rev_dimensions_b b inner join rev_dimensions_tl t on b.dimension_id = t.dimension_id and t.dimension_name like '<dimension name>'

     Replace <dimension name> in the preceding query with the Dimension Name

you find in the UI (Financial Service Application >Master Maintenance > Dimension Management) for the dimension on which the Hierarchy you want to flatten is configured.

- pHierarchyId – Enter Hierarchy id. If all the hierarchies belonging to a dimension are to be processed then, provide NULL as the parameter value. Else, provide the System Identifier of the hierarchy that needs to be transformed.

  Execute the following query in database if only a single hierarchy is to be processed and use the value in hierarchy_id column as parameter for the hierarchy to be processed:

  select b.object_definition_id , short_desc,long_desc from fsi_m_object_definition_b b inner join fsi_m_object_definition_tl t on b.object_definition_id = t.object_definition_id and b.id_type = 5

  Example

  If all the hierarchies for GL Account dimension must be processed, the parameter list should be given as follows (where '2' is the dimension id for the seeded dimension GL Account):

  '2',null

  Example

  If a particular hierarchy with code 1000018112 must be processed (you can obtain this code by executing the preceding query in the database), the parameter list should be given as follows:

  '2', '1000018112'

SQL Example

```
SQL> var fn_return_val number;
SQL> execute :fn_return_val:= rev_batchHierFlatten ('Batch1 ',
'20091231 ', '2 ', '1000018112');
SQL> print fn_return_val
```

PLSQL Example:

```
DECLARE
  fn_return_val number := null;
BEGIN
  fn_return_val := rev_batchHierFlatten('Batch1',
                                        '20091231',
                                         '2',
                                           1000018112');
  IF fn_return_val = 1 THEN

    Dbms_output.put_line('Execution status of batchHierFlatten is'
                         ||fn_return_val || ' --Successful');
  ELSIF fn_return_val = 0 THEN

    Dbms_output.put_line('Execution status of batchHierFlatten is'
                         ||fn_return_val || ' --FAILURE');
          END IF;
EXCEPTION

  WHEN OTHERS THEN
    Dbms_output.put_line('Execution status of batchHierFlatten is'
                         || SQLCODE || '-' || SQLERRM);

END;
```

On successful execution of rev_batchHierFlatten function in Database, value returned will be 1 or 0. 1 indicates successful execution and 0 indicates failure in execution. This function will be present in Atomic Schema.

2. Using OFSAAI ICC Framework

   To execute the procedure from OFSAAI ICC framework, run the batch mentioned below and specify the following parameters:

   - Datastore Type:- Select appropriate datastore from the list
   - Datastore Name:- Select appropriate name from the list
   - IP address:- Select the IP address from the list
   - Rule Name:- *batch_hierTransformation*
   - Parameter List:- Dimension ID, Hierarchy ID

For more information on Hierarchy Transformation, see *Oracle Financial Services Profitability Analytics User Guide*.

Dim Dates Population

DIM_DATES_POPULATION is a seeded Data Transformation which is installed as part of the OFSPA Solution Installer. Time dimension population transformation is used to populate the dim_dates table with values between two dates specified by the user. DIM_DATES table stores the date details required for building OFSPA cubes.

For more information on Time Dimension Population, see *Oracle Financial Services*

Profitability Analytics User Guide.

Fact Ledger Stat Transformation

FSI_LEDGER_STAT_TRM is a seeded Data Transformation which is installed as part of the OFSPA Solution Installer. Fact Ledger Population transformation is used to populate the FCT_LEDGER_STAT table from the Profitability LEDGER_STAT table. Database function LEDGER_STAT_TRM is called by the function FSI_LEDGER_STAT_TRM.

For more information on Fact Ledger Population, refer to *Oracle Financial Services Profitability Analytics User Guide*.

Financial Element Dimension Population

Financial Element Dimension Population involves populating custom Financial Elements created into DIM_FINANCIAL_ELEMENT table from DIM_FINANCIAL ELMENT_B table.

This section covers the following topics:

- Prerequisites
- Tables used by the Financial_Elem_Update transformation
- Executing the Financial_Elem_Update transformation
- Checking the execution status

Prerequisites

1. All the post install steps mentioned in the *Oracle Financial Services Analytical Applications Infrastructure (OFSAAI) Installation and Configuration guide* and the solution installation manuals of *Profitability Management* and *Profitability Analytics* have to be completed successfully.
2. Application User must be mapped to a role that has seeded batch execution function (BATPRO).
3. Seeded and Custom Financial Elements are required to be available in DIM_FINANCIAL_ELEMENTS_B, DIM_FINANCIAL_ELEMENTS_TL tables.
4. Before executing a batch check if the following servers are running on the application server (For more information on how to check if the services are up and on and how to start the services if you find them not running, see *Oracle Financial Services Analytical Applications Infrastructure User Guide*).

1. Iccserver
2. Router
3. AM Server
4. Messageserver
5. Olapdataserver

5. Batches will have to be created for executing the function.

Tables Used by the Financial_Elem_Update Transformation

DIM_FINANCIAL_ELEMENT – This table stores the seeded and custom Financial Elements.

For more details on viewing the structure of the tables, see *OFSPA Erwin Data Model.*

Executing the Financial_Elem_Update Transformation

To execute the function from OFSAAI Information Command Center (ICC) frame work, create a batch by performing the following steps:

> **Note:** For a more comprehensive coverage of configuration and execution of a batch, see *Oracle Financial Services Analytical Applications Infrastructure User Guide*.

1. From the **Home** menu, select **Operations**, then select **Batch Maintenance**.
2. Click **New Batch** ('+' symbol in Batch Name container) and enter the Batch Name and description.
3. Click **Save**.
4. Select the Batch you have created in the earlier step by clicking on the checkbox in the Batch Name container.
5. Click **New Task** ('+' symbol in Task Details container).
6. Enter the Task ID and Description.
7. Select **Transform Data**, from the components list.
8. Select the following from the Dynamic Parameters List and then click **Save**:

- Datastore Type - Select appropriate datastore from the list
- Datastore Name - Select appropriate name from the list
- IP address - Select the IP address from the list
- Rule Name - Select **Financial_Elem_Update** from the list of all available transformations. (This is a seeded Data Transformation which is installed as part of the OFSPA solution installer. If you don't see this in the list, contact Oracle support)
- Parameter List – OFSAAI Application User Name (Refer the following for details on Parameter list)

  Explanation for the parameter list is:

  - Application User Name – This is the OFSAAI application user name which the transformation uses for inserting in DIM_FINANCIAL_ELEMENT table.
  - Sample parameter for this task is 'APPUSER'.

9. Execute the batch.

The function can also be executed directly on the database through SQLPLUS. Details are:

Function Name : fn_dim_financial_elem_update

Parameters : pBatch_Id, pas_of_date, appuser_name

Sample parameter values : 'Batch1','20091231', 'APPUSER'

Checking the Execution Status

The status of execution can be monitored using the batch monitor screen.

> **Note:** For a more comprehensive coverage of configuration & execution of a batch, see *Oracle Financial Services Analytical Applications Infrastructure User Guide*.

The status messages in batch monitor are :

N - Not Started

O - On Going

F - Failure

S – Success

The Event Log window in Batch Monitor provides logs for execution with the top row being the most recent. If there is any error during execution, it will get listed here. Even if you see Successful as the status in Batch Monitor it is advisable to go through the Event Log and re-check if there are any errors. The execution log can be accessed on the application server by going to the following directory $FIC_DB_HOME/log/date. The file name will have the batch execution id.

The database level operations log can be accessed by querying the FSI_MESSAGE_LOG table. The batch run id column can be filtered for identifying the relevant log.

Check the **.profile** file in the installation home if you are not able to find the paths mentioned earlier.

Following messages will be available in the FSI_MESSAGE_LOG table after executing the batch.

1. Starting to update DIM_FINANCIAL_ELEMENT
2. Please provide application user name (In case OFSAAI application user name is not passed as a parameter).
3. Successfully Completed.

After successful execution of the batch, user can verify custom financial element present in DIM_FINANCIAL_ELEMENT table.

Payment Pattern Loader

The Payment Pattern Loader provides the ability to load bulk payment pattern definitions through a back end procedure. This Loader reads the stage table data, does data quality checks on the same, and load them into FSI_PAYMENT_PATTERN and FSI_PAYMENT_PATTERN_EVENT tables, if the stage table data is valid.

Following is the stage table to input the payment pattern:

Table Name: **STG_PAYMENT_PATTERN**

Column Name	Column Datatype	Column Null Option	Column Is PK	Column Comment
V_AMRT_TYPE	VARCHAR2(5)	NOT NULL	Yes	Amortization code between 1000 to 69999. Patterns between this range will be consider for payment processing
N_EVENT_ID	NUMBER(5,0)	NOT NULL	Yes	Event Identity Number
N_SPLIT_ID	NUMBER(5,0)	NOT NULL	Yes	Holds number of patterns with in split pattern
V_PATTERN_TYPE	VARCHAR2(40)	NOT NULL	Yes	List of values could be Absolute, Relative, Split
V_TERM_TYPE	VARCHAR2(40)	NOT NULL	Yes	List of values could be Principal and Interest, Principal Only, Interest Only, Level Principal, Final Principal & Interest, Other
V_AMRT_TYPE_DESC	VARCHAR2(255)	NOT NULL	No	Alpha numeric value
N_PCT_VALUE	NUMBER(8,4)	NULL	No	Percentage applied to each pattern in case of split pattern type
V_PAYMENT_EVENT_MONTH	VARCHAR2(20)	NULL	No	Month in which payment event should occur

Column Name	Column Datatype	Column Null Option	Column Is PK	Column Comment
N_PAYMENT_EVENT_DAY	NUMBER(2)	NULL	No	Number of Days Payment type
N_PAYMENT_EVENT_FREQ	NUMBER(5,0)	NULL	No	Number of times payment event should occur
V_PAYMENT_EVENT_FREQ_MULT	VARCHAR2(40)	NULL	No	List of values could be Days,Months,Years
N_PAYMENT_EVENT_REPEAT_VALUE	NUMBER(5,0)	NULL	No	Holds number of times payment frequency should repeat
N_AMOUNT	NUMBER(14,2)	NULL	No	Amount
V_AMOUNT_TYPE	VARCHAR2(40)	NULL	No	List of values could be % of original Payment,% of current payment,absolute value
V_PAYMENT_TYPE	VARCHAR2(30)	NULL	No	List of values could be Conventional, Level principal, Non-amortizing

The loader program performs the following data quality checks:

1. The below fields, list of values will be checked against the relevant look tables as mentioned

   Pattern Type ---> FSI_PATTERN_TYPE_MLS

   CashFlowType ---> FSI_PAYMENT_TYPE_MLS (Should accept only "100-Principal and Interest" and "300-Interest Only")

   Month ---> FSI_MONTHS_MLS

Multiplier ---> FSI_MULTIPLIER_MLS

Payment Method ---> FSI_AMOUNT_TYPE_MLS (For Conventional accept only "% of Original Payment", "% of Current Payment" and "Absolute value")

Payment Type ---> FSI_PMT_PATTERN_TYPE_MLS

2. While defining any pattern type like relative or absolute, MONTH and DAY combination should be unique

3. MONTH and DAY pair should have valid month and day combination, such as January 31 days and February 28 days (Leap year was not considered) and so on.

4. If cash flow value is "Principal and Interest" then N_AMOUNT cannot be blank. If it is "Interest only" then V_PAYMENT_TYPE and N_AMOUNT should be blank.

5. When payment type is a Non-amortizing and Payment pattern is Relative then N_PAYMENT_EVENT_FREQ and N_PAYMENT_EVENT_REPEAT_VALUE should have values which could range between 1 to 9999.

6. One Split pattern can have any number of definition, however the sum of N_PCT_VALUE of all the definition should be 100% and all the payment patterns in the split should be defined.

7. All the following fields should have this validation on place:

   Day ---> Positive Integer Number Range from 1 to 31 depends on the month for which day.

   Percentage ---> Positive Integer or Decimal Number

   Frequency ---> Positive Integer range from 1 to 9999

   Repeat ---> Positive Integer range from 1 to 9999

   Value ---> Integer numbers from 0 to 9999999999

8. For each payment pattern and payment type combinations fields relevant to that would be populated by the user remaining columns should be populated with default values:

   - **Payment Pattern - Absolute**
     - Payment Type: Conventional / Level Principal
     - Columns gets populated with user values: Code , Description, Pattern Type, Payment Type, Month, Day, Cash Flow Type, Payment Method,Value, Percentage (in case of Split pattern type)
     - Payment Type: Non amortizing Payment type.

- Columns gets populated with user values: Code , Description , Pattern Type, Payment Type, Month, Day, Percentage (in case of Split pattern type)

- **Payment Pattern - Relative**

- Payment Type: Conventional / Level Principal

- Columns gets populated with user values: Code, Description, Pattern Type, Payment Type, Frequency, Multiplier, Repeat, Cash Flow Type, Payment Method, Value, Percentage (in case of Split pattern type)

- Payment Type: Non amortizing Payment type.

- Columns gets populated with user values: Code, Description, Pattern Type, Payment Type, Frequency, Multiplier, Repeat, Percentage (in case of Split pattern type)

Note: The loader program defaults values for each column in case values provided by user are not relevant for the pattern and payment patterns they defined.

Table Name	Column Name	Default Value
FSI_PAYMENT_PATTERN_EVENT	EVENT_ID	0
FSI_PAYMENT_PATTERN_EVENT	TERM_TYPE_CD	0
FSI_PAYMENT_PATTERN	PERCENTAGE	0
FSI_PAYMENT_PATTERN_EVENT	PAYMENT_EVENT_MONTH	0
FSI_PAYMENT_PATTERN_EVENT	PAYMENT_EVENT_DAY	0
FSI_PAYMENT_PATTERN_EVENT	PAYMENT_EVENT_FREQUENCY	0
FSI_PAYMENT_PATTERN_EVENT	PAYMENT_EVENT_FREQUENCY_MULT	M
FSI_PAYMENT_PATTERN_EVENT	PAYMENT_EVENT_REPEAT	0
FSI_PAYMENT_PATTERN_EVENT	AMOUNT	0

Table Name	Column Name	Default Value
FSI_PAYMENT_PATTERN_EVENT	AMOUNT_TYPE_CD	0
FSI_PAYMENT_PATTERN	SPLIT_ID	0
FSI_PAYMENT_PATTERN_EVENT	N_SPLIT_ID	0

Executing the Payment Pattern Loader Procedure

There are two ways to execute the Payment Pattern Loader procedure:

- **Running Procedure Using SQL*Plus**

  To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

```
SQLPLUS > declare
result number;
begin
result := fn_paymentpattern
('SimpleDIIM_BATCH1','20121212','730','Y');
end;
/
```

  - BATCH_RUN_ID is any string to identify the executed batch.
  - AS_OF_DATE is in the format YYYYMMDD.

- **Payment Pattern Loader Procedure Using OFSAAI ICC Framework**

  To execute Payment Pattern Loader from OFSAAI ICC framework, a seeded Batch is provided.

  The batch parameters are:

  - Datastore Type:- Select the appropriate datastore from list.
  - Datastore Name:- Select the appropriate name from the list.
  - IP address:- Select the IP address from the list.
  - Rule Name:- fn_paymentpattern

Exception Messages

Below are the list of error messages which can be viewed in view log from UI or fsi_message_log table from back end filtering for the given batch id. On successful completion of each task messages gets into log table.

In the event of failure, following are the list of errors that may occur during the execution:

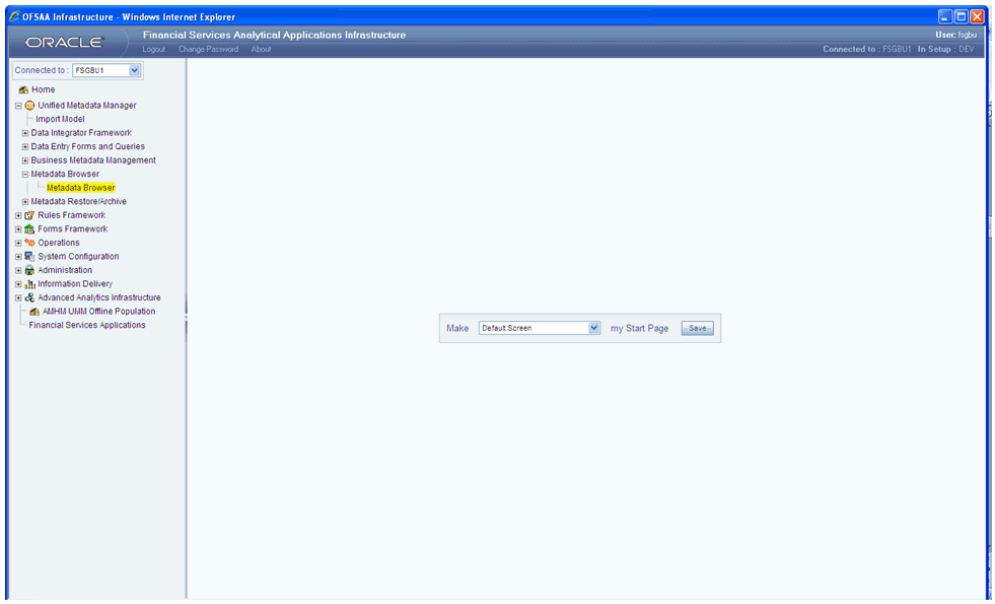
- **Exception 1:** PATTERN TYPE IS NOT MATCHING THE LIST OF VALUES
- **Exception 2:** TERM TYPE IS NOT MATCHING THE LIST OF VALUES
- **Exception 3:** PAYMENT MULTIPLIER FREQ IS NOT MATCHING THE LIST OF VALUES
- **Exception 4:** EVENT MONTH IS NOT MATCHING THE LIST OF VALUES
- **Exception 5:** PMT PATTERN IS NOT MATCHING THE LIST OF VALUES
- **Exception 6:** AMOUNT TYPE IS NOT MATCHING THE LIST OF VALUES
- **Exception 7:** v_amrt_type_cd range between 1000 to 69999
- **Exception 8:** n_payment_event_freq_cd and n_payment_event_repeat_value should have values range between 1 to 9999
- **Exception 9:** N_PCT_VALUE POSITIVE INTEGER OR DECIMAL NUMBER
- **Exception 10:** N_SPLIT_ID POSITIVE NUMBER
- **Exception 11:** If Term Type is PRINCIPAL AND INTEREST, then it should range from 1 to 999999999
- **Exception 12:** f Term Type is INTEREST ONLY AMOUNT and AMOUNT TYPE CD should be blank
- **Exception 13:** ERR while deleting stg_payment_pattern unwanted records
- **Exception 14:** Month and Day pair should have valid combination. Like January 31 days
- **Exception 15:** Pct value should be 100 in case of split pattern for other patterns it should be =0
- **Exception 16:** Error during percentage check
- **Exception 17:** Error during inserting

5

Mapping Export in Metadata Browser

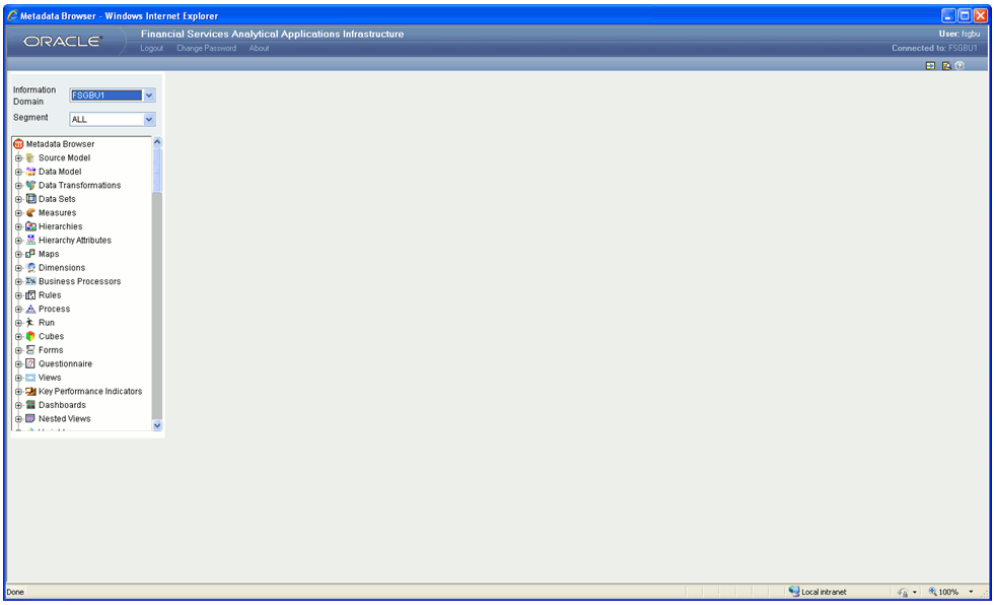
Procedure

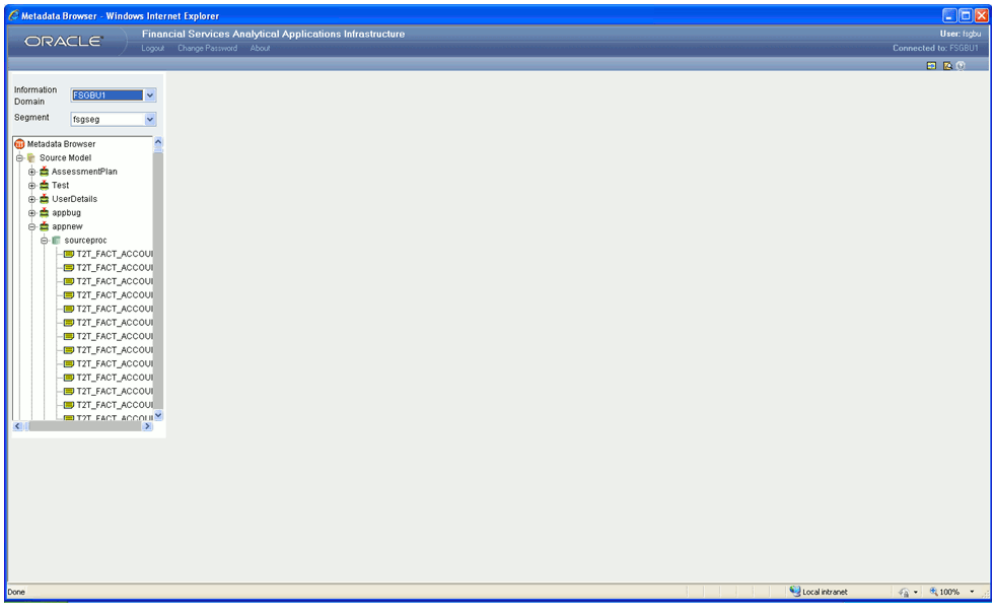
1. Login to OFSAAI Screen.
2. Click **Unified Metadata Manager**. Then, navigate to **Metadata Browser**.

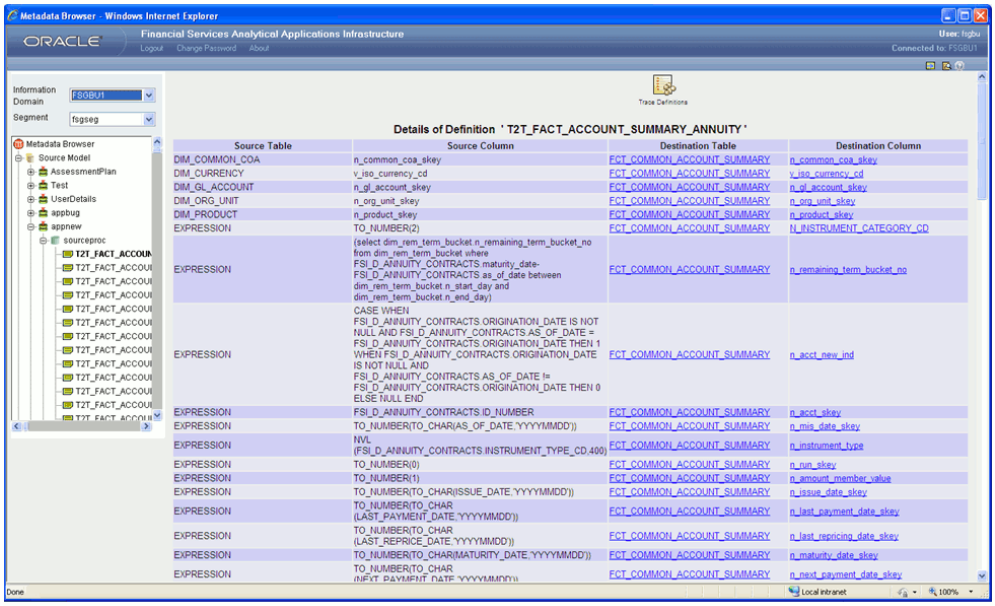


   Metadata Browser window will open.

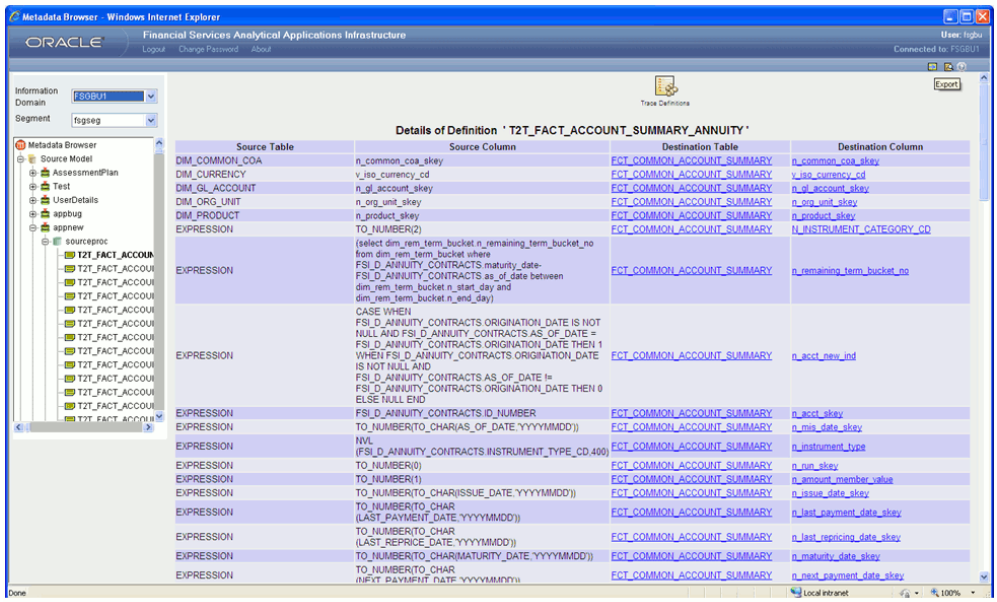
3. Select the correct Segment from the dropdown.
4. Then select the corresponding Source Model as shown in the below screenshots.


Metadata Browser - Windows Internet Explorer
ORACLE
Financial Services Analytical Applications Infrastructure
Logout Change Password About
User: fsgbu
Connected to: FSGBU1
Information Domain
FSGBU1
Segment
ALL
Metadata Browser
Source Model
Data Model
Data Transformations
Data Sets
Measures
Hierarchies
Hierarchy Attributes
Maps
Dimensions
Business Processors
Rules
Process
Run
Cubes
Forms
Questionnaire
Views
Key Performance Indicators
Dashboards
Nested Views
Done
Local intranet
100%

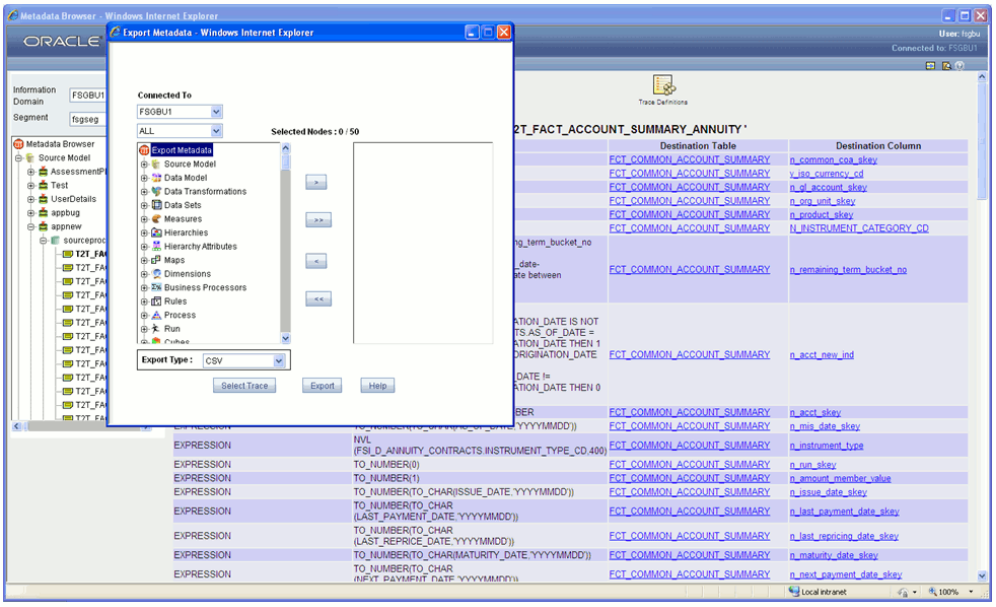

Metadata Browser - Windows Internet Explorer
ORACLE
Financial Services Analytical Applications Infrastructure
Logout Change Password About
User: fsgbu
Connected to: FSGBU1
Information Domain
FSGBU1
Segment
fsgseg
Metadata Browser
Source Model
AssessmentPlan
Test
UserDetails
appbug
appnew
sourceproc
T2T_FACT_ACCOUI
Done
Local intranet
100%



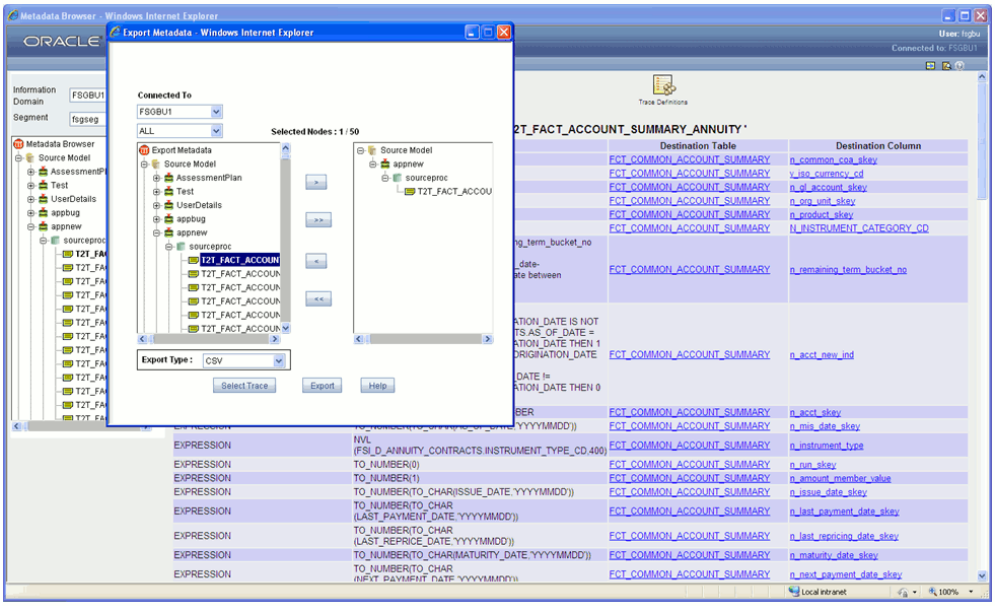
5. In the Top Banner , select the **Export** button.



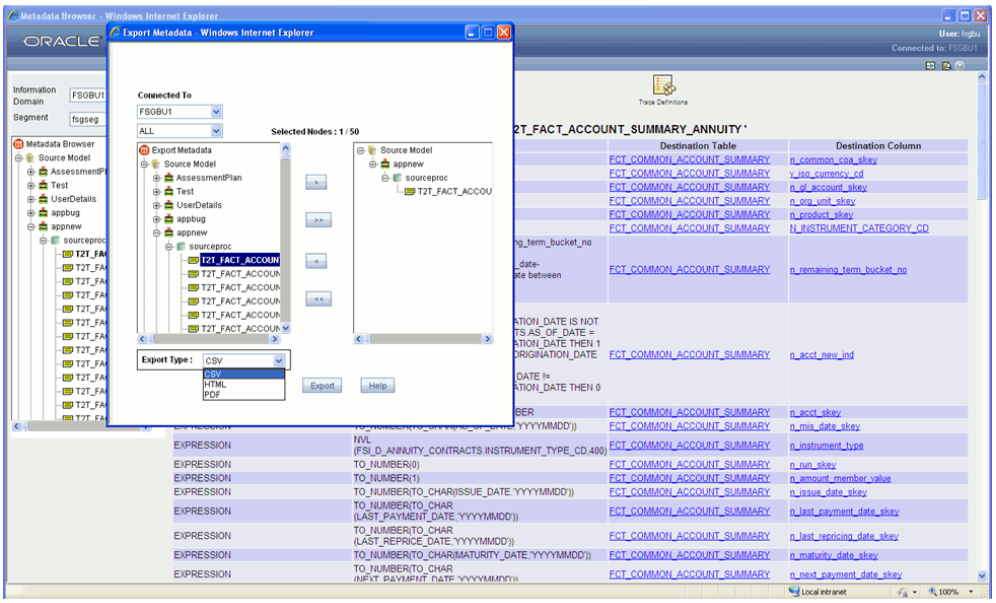
6. Export window will open as shown in the below screenshots.



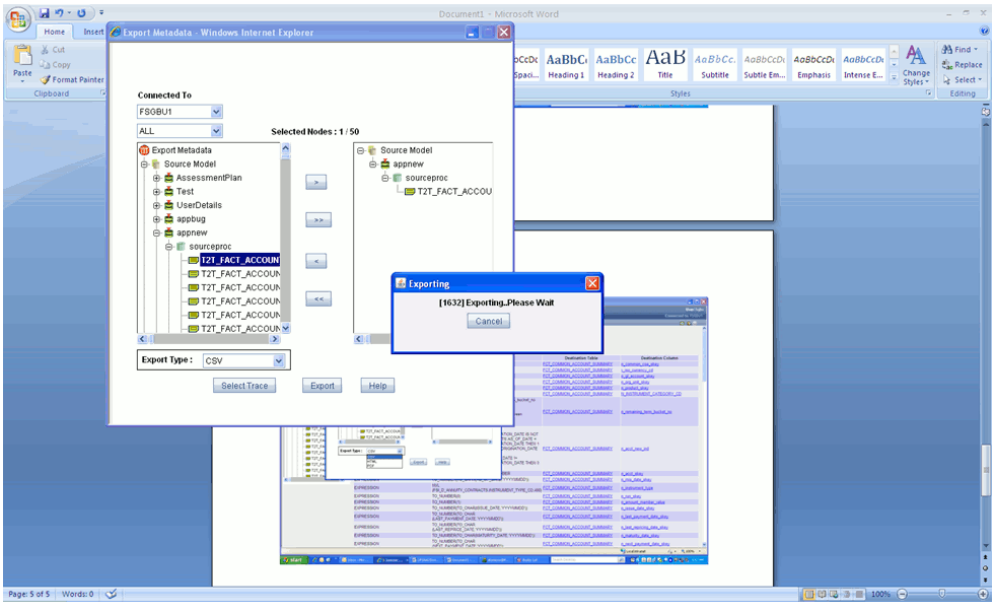
7. Again select the same Source Model and map to the Right Hand Side as shown below for exporting.



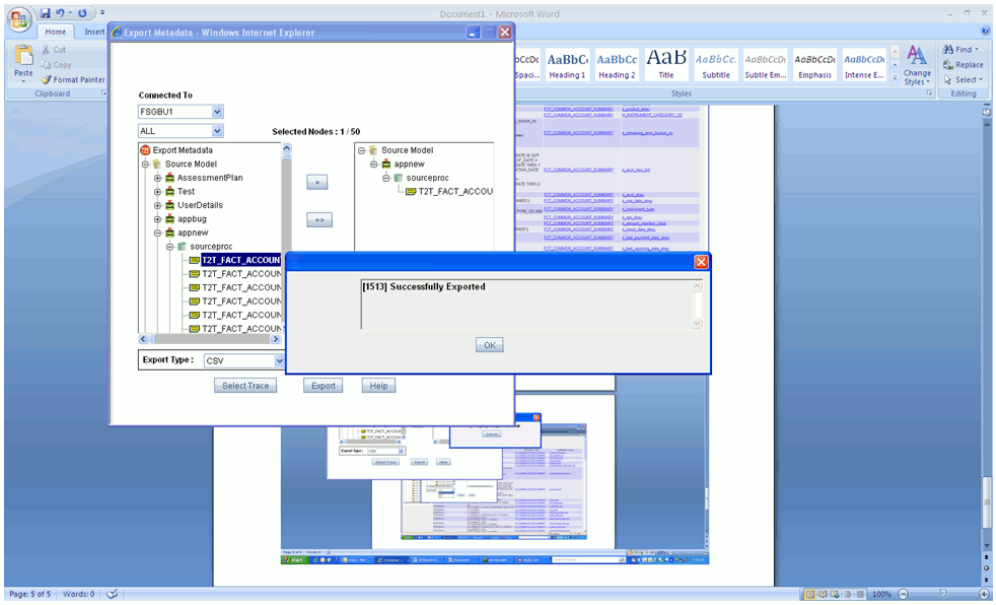
8. Select the **Export Type.** Click **Export.**



9. **Export is progressing** message is displayed.



10. **Successfully Exported** message is displayed. Click **Ok.**.


Export Metadata - Windows Internet Explorer
Connected To
FSGBU1
ALL
Selected Nodes : 1 / 50
Export Metadata
Source Model
AssessmentPlan
Test
UserDetails
appbug
appnew
sourceproc
T2T_FACT_ACCOUN
Export Type :
CSV
Select Trace
Export
Help
[1513] Successfully Exported
OK

6

SCD Configuration

Overview of SCD Process

SCDs are dimensions that have data that changes slowly, rather than changing on a time-based, regular schedule.

For more information on SCDs, refer to:

- *Oracle Data Integrator Best Practices for a Data Warehouse* at

  <http://www.oracle.com/technetwork/middleware/data-integrator/overview/odi-bestpractices-datawarehouse-whi-129686.pdf >

- *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* at

  <http://download.oracle.com/docs/cd/E16338_01/owb.112/e10935/dim_objects.htm>

Additional online sources include:

- <http://en.wikipedia.org/wiki/Slowly_changing_dimension>
- <http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/10g/r2/owb/owb10gr2_gs/owb/lesson3/slowlychangingdimensions.htm>
- <http://www.oraclebidwh.com/2008/11/slowly-changing-dimension-scd/>
- <http://www.informationweek.com/news/software/bi/showArticle.jhtml?articleID=204800027&pgno=1>
- <http://www.informationweek.com/news/software/bi/showArticle.jhtml?articleID=59301280>

You can also refer to *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling* by Ralph Kimball and Margy Ross.

The SCD component of the platform is delivered via a C++ executable. The types of SCD

handled by the OFSAAI SCD component for OFSPA solution are Type 1 and Type 2.

Type 1

The Type 1 methodology overwrites old data with new data, and therefore does not track historical data. This is useful for making changes to dimension data.

Example

N_PRODUCT_SKEY	V_PRODUCT_NAME	D_START_DATE	D_END_DATE	F_LATEST_RECORD_INDICATOR
1	PL	5/31/2010	12/31/9999	Y

In this example,

N_PRODUCT_SKEY is the surrogate key column which is a unique key for each record in the dimension table.

V_PRODUCT_NAME is the product name.

D_START_DATE indicates the date from which this product record is valid.

D_END_DATE indicates the date till which this product record is valid.

F_LATEST_RECORD_INDICATOR with value 'Y', which indicates this is the latest record in the dimension table for this product and 'N' indicates it is not.

If the V_PRODUCT_NAME column is set as a Type 1 SCD column and if there is a change in the product name to 'Personal Loan' from 'PL' in the above example, in the next processing period, then when SCD is executed for the new processing period the record in the above example changes to:

N_PRODUCT_SKEY	V_PRODUCT_NAME	D_START_DATE	D_END_DATE	F_LATEST_RECORD_INDICATOR
1	Personal Loan	6/30/2010	12/31/9999	Y

Type 2

The Type 2 method tracks historical data by creating multiple records for a given natural key in the dimensional tables with separate surrogate keys. With Type 2, the historical changes in dimensional data are preserved. In the above example for the change in product name from 'PL' to 'Personal Loan' if history has to be preserved, then the V_PRODUCT_NAME column has to be set as Type 2 when SCD is processed for the processing period and the change inserts a new record as shown in the following example:

Example

N_PRODUCT_SKEY	V_PRODUCT_NAME	D_START_DATE	D_END_DATE	F_LATEST_RECORD_INDICATOR
1	PL	5/31/2010	12/31/9999	N
1	Personal Loan	6/30/2010	12/31/9999	Y

A new record is inserted to the product dimension table with the new product name. The latest record indicator for this is set as 'Y', indicating this is the latest record for the personal loan product. The same flag for the earlier record was set to 'N'.

Prerequisites

1. The Hierarchy Flattening Transformation should have been executed successfully.
2. The SCD executable should be present under *<installation home>ficdb/bin*. The file name is **scd** and the user executing the SCD component should have execute rights on this file.
3. The setup tables accessed by SCD component (SETUP_MASTER, SYS_TBL_MASTER, and SYS_STG_JOIN_MASTER) should have the required entries. The SETUP_MASTER table does not come seeded with the installation; the required entries must be added manually. The required columns are mentioned in the Tables Used by the SCD Component, . The tables SYS_TBL_MASTER and SYS_STG_JOIN_MASTER are seeded for the Org unit, GL Account, Product, Common COA (Chart of Accounts) dimensions along with solution installation and you must only add entries in these tables, if you add new dimensions.
4. Database Views with name DIM_<Dimension Name>_V come seeded, for the seeded dimensions which come as part of installation. These views source data from the Profitability dimension tables as well as the flattened hierarchy data.

   DIM_PRODUCT_V is the view available for the product dimension.

   New views will have to be added for any new dimension, added in addition to the seeded dimensions.

Tables Used by the SCD Component

The following are the database tables and columns used by the SCD component:

- SETUP_MASTER
  - V_COMPONENT_CODE - This column is not used by the OFSPA solution.

- V_COMPONENT_DESC - This column value is hard coded in the database view definitions for DIM_PRODUCT_V, DIM_GL_ACCOUNT_V, DIM_COMMON_COA_V, and DIM_ORG_UNIT_V to obtain the Hierarchy ID from the REV_HIER_FLATTENED table. For this reason, the value for this column should be unique.

  Note: The value in V_COMPONENT_DESC must exactly match with the value used in the SQL to create the DIM_<dimension>_V view. The View SQL contains a section referencing the SETUP_MASTER table. You must use the same upper and/or lower case letters in V_COMPONENT_DESC as used in this section of the View SQL.

- V_COMPONENT_VALUE - This is the hierarchy ID to be processed and this can be obtained by executing the following query:

```
select b.object_definition_id,short_desc,long_desc from
fsi_m_object_definition_b b inner join fsi_m_object_definition_tl
t on b.object_definition_id = t.object_definition_id and
b.id_type = 5
```

Example:

V_COMPONENT_CODE	V_COMPONENT_DESC	V_COMPONENT_VALUE
COMMON_COA_HIER	COMMON_COA_HIER1	1000063952
GL_ACCOUNT_HIER	GL_ACCOUNT_HIER1	200000808
ORG_HIER	ORG_UNIT_HIER1	200282
PRODUCT_HIER	PRODUCT_HIER1	1000004330

Note: *For any newly defined Hierarchy, a row will have to be inserted to this table manually for SCD to process that Hierarchy. You can only specify one Hierarchy for each dimension.*

- SYS_TBL_MASTER

  The solution installer populates one row per dimension for the seeded dimensions in this table.

Column Name	Data Type	Column Description
MAP_REF_NUM	NUMBER(3) NOT NULL	The Mapping Reference Number for this unique mapping of a Source to a Dimension Table.
TBL_NM	VARCHAR2(30) NOT NULL	Dimension Table Name.
STG_TBL_NM	VARCHAR2(30) NOT NULL	Staging Table Name.
SRC_PRTY	NUMBER(2) NULL	Priority of the Source when multiple sources are mapped to the same target.
SRC_PROC_SEQ	NUMBER(2) NOT NULL	The sequence in which the various sources for the DIMENSION will be taken up for processing.
SRC_TYP	VARCHAR2(30) NULL	The type of the Source for a Dimension, that is, Transaction Or Master Source.
DT_OFFSET	NUMBER(2) NULL	The offset for calculating the Start Date based on the Functional Requirements Document (FRD).
SRC_KEY	NUMBER(3) NULL	

Example:

This is the row inserted by the solution installer for the product dimension.

MAP_REF_NUM	128

TBL_NM	DIM_PRODUCT
STG_TBL_NM	DIM_PRODUCT_V
SRC_PRTY	
SRC_PROC_SEQ	1
SRC_TYP	MASTER
DT_OFFSET	0

Note: *For any newly defined dimension, a row will have to be inserted to this table manually.*

- SYS_STG_JOIN_MASTER

  The solution installer populates this table for the seeded dimensions.

Column Name	Data Type	Column Description
MAP_REF_NUM	NUMBER(3) NOT NULL	The Mapping Reference Number for this unique mapping of a Source to a Dimension Table.
COL_NM	VARCHAR2(30) NOT NULL	Name of the column in the Dimension Table.
COL_TYP	VARCHAR2(30) NOT NULL	Type of column. The possible values are given in the following section.
STG_COL_NM	VARCHAR2(60) NULL	Name of the column in the Staging Table.
SCD_TYP_ID	NUMBER(3) NULL	SCD type for the column.

Column Name	Data Type	Column Description
PRTY_LOOKUP_REQD_FLG	CHAR(1) NULL	Column to determine whether Lookup is required for Priority of Source against the Source Key Column or not.
COL_DATATYPE	VARCHAR2(15) NULL	The list of possible values are VARCHAR, DATE, and NUMBER, based on the underlying column datatype.
COL_FORMAT	VARCHAR2(15) NULL	

The possible values for column type (the COL_TYPE column) in SYS_STG_JOIN_MASTER table are:

1. PK - Primary Dimension Value (can be the multiple of the given *Mapping Reference Number*)
2. SK - Surrogate Key
3. DA - Dimensional Attribute (may be multiple for a given "Mapping Reference Number")
4. SD - Start Date
5. ED - End Date
6. LRI - Latest Record Indicator (Current Flag)
7. CSK - Current Surrogate Key
8. PSK - Previous Surrogate Key
9. SS - Source Key
10. LUD - Last Updated Date/Time
11. LUB - Last Updated By

Example:

This is the row inserted by the solution installer for the product dimension.

MAP_REF_NUM	128
COL_NM	V_PRODUCT_NAME
COL_TYP	DA
STG_COL_NM	V_PRODUCT_NAME
SCD_TYP_ID	2
PRTY_LOOKUP_REQD_FLG	N
COL_DATATYPE	VARCHAR
COL_FORMAT	

> **Note:** *For any newly defined dimension, the column details will have to be inserted to this table manually.*

- DIM_< dimension name >_V - The database view which SCD uses as the source.

  Example
  DIM_PRODUCTS_V

  These views come as part of install for the dimensions seeded with the application.

  > **Note:** *For any newly defined dimension, a view will have to be created, which is similar to that of DIM_PRODUCTS_V.*

A sequence should be created for every user-defined dimension, using the below query:

Example

```
create sequence SEQ_< DIMENSION > minvalue 1
maxvalue 999999999999999999999999999
increment by 1
```

Executing the SCD Component

You can execute the SCD component from the Operations (formerly Information Command Center (ICC) framework) module of OFSAAI.

The SCD component for OFSPA 6.1 has been seeded with the Batch ID **<INFODOM>_SCD**, which can be executed from *Batch Execution* section of OFSAAI.

You can also define a new Batch and an underlying Task definition from the *Batch Maintenance* window of OFSAAI. For more information on defining a new Batch, refer to section How to Define a Batch, .

To define a new task for a Batch definition:

- Select the check box adjacent to the newly created Batch Name in the *Batch Maintenance* window.
- Click **Add** (+) button from the Task Details grid.

  The *Task Definition* window is displayed.
- Enter the **Task ID** and **Description**.
- Select **Run Executable** component from the drop down list.
- Select the following from the **Dynamic Parameters** list:
  - **Datastore Type** - Select the appropriate datastore type from the list
  - **Datastore Name** - Select the appropriate datastore name from the list
  - **Executable** - Enter **scd,<map ref num>**

    For example, **scd,2**
  - **Wait** - Click **Yes** if you want to wait till the execution is complete or click **No** to proceed with the next task.

    Important: Always select **N** in Wait Parameter.
  - **Batch Parameter** - Click **Yes** in Batch Parameter field if you want to pass the batch parameters to the executable and click NO otherwise.

    Important: Always select **Y** in Batch Parameter.
- Click **Save**.

  The Task definition is saved for the selected Batch.
- Click **Parameters.** Select the following from the Dynamic Parameters List and then click **Save**:

  The map ref number values available for the **Executable** parameter are:
  - **-1**, if you want to process all the dimensions. The *Executable* parameter mentioned earlier is:

scd,-1.

- If you want to process for a single dimension, query the database table SYS_TBL_MASTER and give the number in the MAP_REF_NUM column for the dimension you want to process. These are the ones which come seeded with the install. If you want to process for Product dimension, the *Executable* parameter mentioned earlier is:

  scd,6.

MAP_REF_NUM	TBL_NM
126	DIM_ORG_UNIT
127	DIM_GL_ACCOUNT
128	DIM_PRODUCT
129	DIM_COMMON_COA

- You can execute a Batch definition from the *Batch Execution* section of *OFSAAI Operations* module.

Checking the Execution Status

The Batch execution status can be monitored through *Batch Monitor* section of *OFSAAI Operations* module.

The status messages in batch monitor are:

N - Not Started

O - On Going

F - Failure

S – Success

The execution log can also be accessed on the application server in the directory *$FIC_DB_HOME/log/ficgen*, where file name will have the Batch Execution ID.

The detailed SCD component log can be accessed on the application server in the directory $FIC_HOME by accessing the following path */ftpshare/<infodom name>/logs*.

> **Note:** Check the **.profile** file in the installation home if you are unable to find this path.

The *Event Log* window in *Batch Monitor* section provides execution logs, in which the top row is the most recent. Any errors during the Batch execution are listed in the logs.