Oracle® Retail Analytics and Planning

Implementation Guide

Release 25.1.101.0
G25188-03
March 2025

Oracle Retail Analytics and Planning Implementation Guide, Release 25.1.101.0

G25188-03

Contents

4 Integration with Merchandising

5 Batch Orchestration

7 Implementation Tools

8 Data File Generation

9 Extensibility

Send Us Your Comments

Oracle Retail Analytics and Planning Implementation Guide

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).

> **Note:**
>
> Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@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 `http://www.oracle.com`.

Preface

This Implementation Guide provides critical information about the processing and operating details of the Analytics and Planning, including the following:

- System configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

This guide is for:

- Systems administration and operations personnel
- System analysts
- Integrators and implementers
- Business analysts who need information about Analytics and Planning processes and interfaces

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 that have purchased support 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.

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Oracle Help Center (docs.oracle.com)

Oracle Retail Product documentation is available on the following website https://docs.oracle.com/en/industries/retail/html

Comments and Suggestions

Please give us feedback about Oracle Retail Help and Guides. You can send an e-mail to: retail-doc_us@oracle.com

Oracle Retail Cloud Services and Business Agility

Oracle Retail Analytics and Planning Cloud Service is hosted in the Oracle Cloud with the security features inherent to Oracle technology and a robust data center classification, providing significant uptime. The Oracle Cloud team is responsible for installing, monitoring, patching, and upgrading retail software.

Included in the service is continuous technical support, access to software feature enhancements, hardware upgrades, and disaster recovery. The Cloud Service model helps to free customer IT resources from the need to perform these tasks, giving retailers greater business agility to respond to changing technologies and to perform more value-added tasks focused on business processes and innovation.

Oracle Retail Software Cloud Service is acquired exclusively through a subscription service (SaaS) model. This shifts funding from a capital investment in software to an operational expense. Subscription-based pricing for retail applications offers flexibility and cost effectiveness.

1 Introduction

Overview

The Oracle Retail Analytics and Planning platform is the common and extensible cloud architecture for analytics and planning solutions.

The platform supports Oracle Retail applications across each of major analytical categories, including:

- Descriptive and diagnostic with merchandise, customer, and consumer insights.
- Predictive with demand forecasting, customer, and location clustering.
- Prescriptive with assortment, pricing, and inventory optimization.

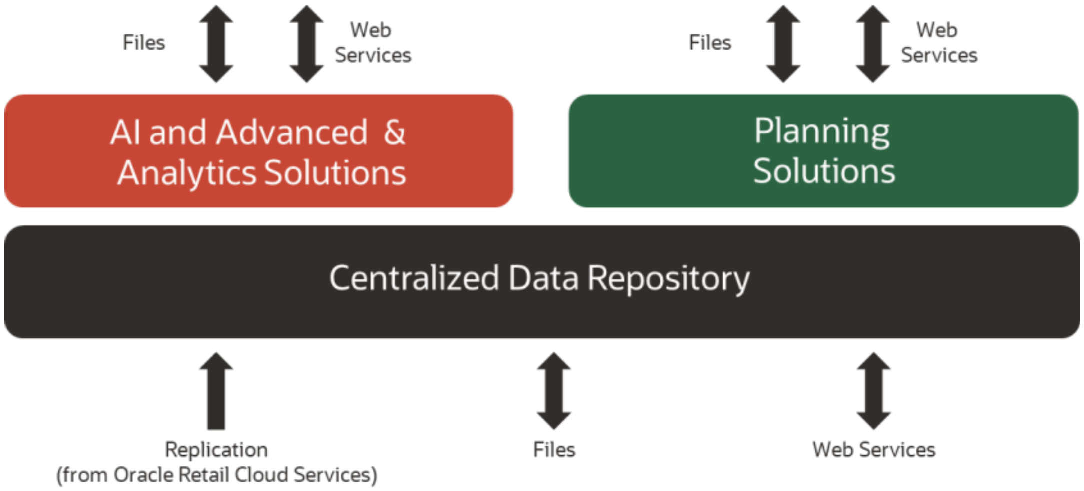
The platform also supports Oracle Retail merchandise and inventory planning solutions. These solutions support business responsiveness through a highly interactive user experience and drive the best outcomes with the application of advanced analytics and artificial intelligence (AI).

As a common platform, it provides a centralized data repository, lean integration APIs, and an efficient portfolio of delivery technologies. The data repository reflects a comprehensive data model of retail planning, operations, and execution processes. The integration APIs support right-time interactions: a lean set of bulk, on-demand, and near real-time mechanisms. The delivery technologies represent a portfolio of connected tools to build and extend composite solutions using fit-for-purpose analytical, application, and integration tools.

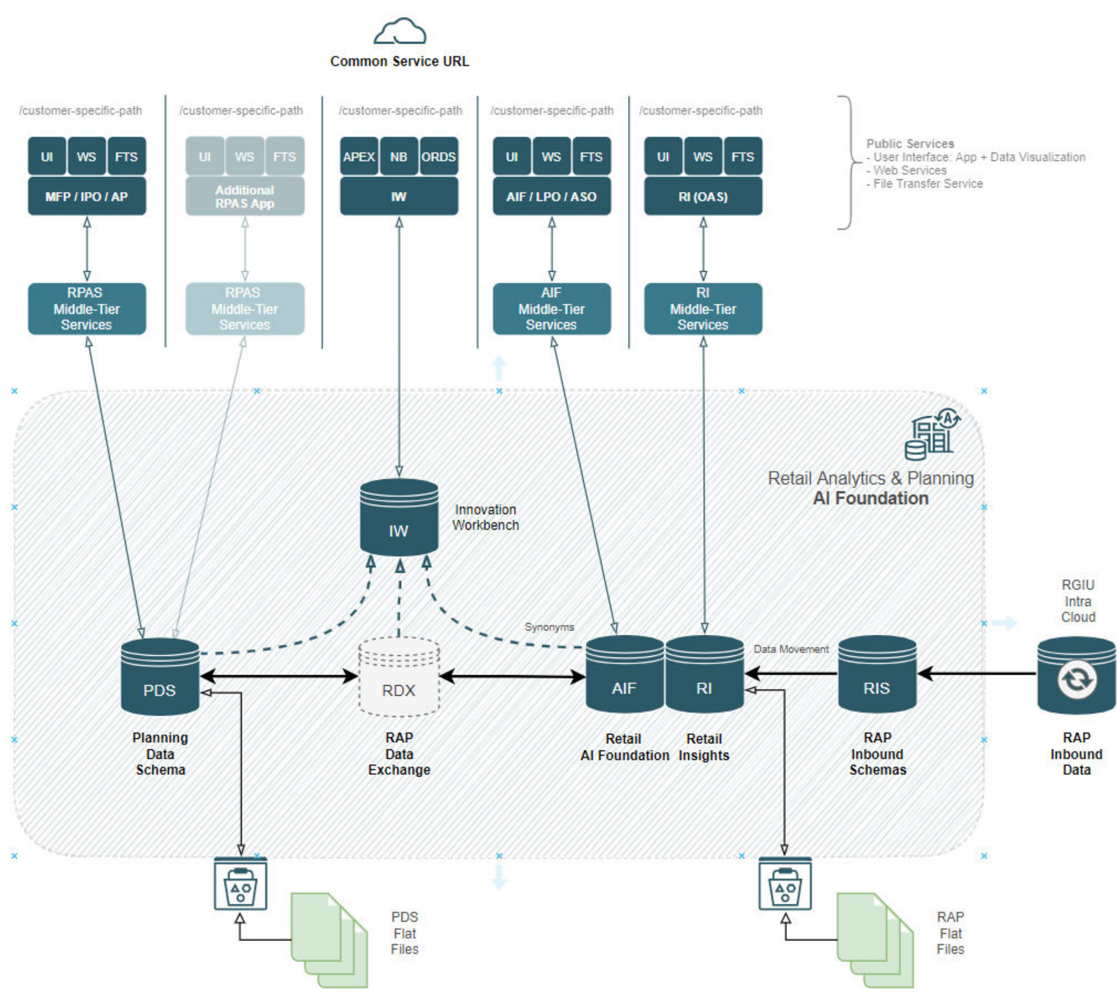
Architecture

The architecture used for Oracle Retail Analytics and Planning provides a centralized repository and integration path for all common foundational and analytical data. The centralized repository ensures that all solutions reference consistent data definitions across transformations and aggregations. The centralized integrations simplify implementation and operational support. This centralization can also be complemented by data and integrations for customer-specific extensions to any analytics or planning solution implemented on the platform. Coordination of analytical processes and data movement within the platform is managed by Oracle Retail Process Orchestration & Monitoring using a common schedule.

The diagram below depicts the high-level platform architecture.



Looking inside the data repository, the internal architecture consists of a single Autonomous Data Warehouse (ADW) instance per customer containing multiple application schemas for each of the RI, AIF, and Planning applications.



These schemas are broadly categorized as follows:

1. RAP Inbound Schemas (RIS) hold copies of Merchandising Foundation CS (MFCS) and/or Customer Engagement (CE) database schemas, replicated in near real-time. Each source application writes to its own destination schema, such as `MFCS_RDS` and `CE_RDS`. The naming conventions are the same as the Retail Data Store (RDS) but these are not the same objects: it is a separate data flow directly to RAP.
2. RAP foundation data coming from external systems (or older versions of MFCS / CE) is pushed into Object Storage using File Transfer Services. If Planning solutions are used with legacy file formats from older versions, that planning data is loaded into a separate Object Storage access point and then loaded directly to PDS.
3. All RAP inbound data passes through the data warehouse, which supports Retail Insights (RI) and AI Foundation (AIF) applications. The primary schemas used are `RADM01` and `RASE01` for RI and AIF, respectively. RI and AIF are tightly coupled and read from each other's schemas in their batch programs.
4. RI and AIF applications exchange data with Planning applications using an intermediate schema called the RAP data exchange (RDX), which handles the data transformations and mappings between the applications. The RDX schema is necessary to allow for Planning extensibility, which can change how data is imported to PDS to fit each retailer's requirements.
5. PDS is the common application schema used by all Planning applications. Data in PDS will come either from RDX or from external files loaded through Object Storage.
6. Innovation Workbench resides in a separate database schema to provide a custom workspace that has read/write access. The default schema name is `RTLWSP01`. IW has access to the other schemas in RAP using a combination of synonyms and grants to specific objects. It does not have unrestricted access to application data; it can only access what has been granted from each application to the IW workspace.

Getting Started

Each implementation of a Retail Analytics and Planning solution involves one or more modules across Insights, AI Foundation, and Planning. It often includes multiple years of historical data sourced from multiple Oracle and non-Oracle systems, some of which will also need to be integrated with the platform on an ongoing basis. For many of the modules, you will also want data from the platform to be sent to other downstream applications and processes. For these reasons, every implementation is unique to your business requirements and requires careful planning and a deep understanding of all platform components.

Regardless of the modules being implemented, the outline in the table below can be followed and adapted for your project, and later chapters of the document will elaborate on many of these topics. More detailed checklists and project planning tools for some modules are also available in My Oracle Support.

Table 1-1 Implementation Outline

Project Phase	Activity	References
Pre-Implementation	Train team members on platform tools, including Retail Home, POM, APEX, Object Store, and Oracle Analytics Server (OAS).	Read the section on Implementation Tools
	Plan for the types and volumes of historical data you will need to create, based on platform and application-specific data needs.	Read the section on Data Requirements

Table 1-1 (Cont.) Implementation Outline

Project Phase	Activity	References
	Plan for all inbound data sources that must provide history and ongoing data to the platform, such as external merchandising systems.	Read the section on Loading Data from Files
	Understand how data is manipulated and transformed to meet the needs of the platform modules, as this can influence your data conversion efforts.	Read the chapter on Data Processing and Transformations
Environment Provisioning	Complete all activities captured in the Service Administrator Action List before starting the implementation.	Oracle Retail Analytics and Planning Service Administrator Action List
	Perform the initial system setup and confirm all configurations with the customer and implementation teams.	Complete the sections on Platform Configurations and Application Configurations
Data Conversion	Extract historical data files based on the needs of your chosen RAP modules.	Complete the section on Data Requirements
	Understand how to interface with RAP to upload and move data into the platform and through to all implemented modules.	Complete the section on Preparing to Load Data
Data Loading	Initialize your system and business calendars and validate that the environment is ready to run batch processes.	Complete the section on Calendar Setup
	Integrate all data files for history loads and initial seeding of positional data.	Complete the sections on Loading Data from Files and Integration with Merchandising
	Ensure all implemented modules have received the necessary historical data for the entire history window.	Complete the sections on Sending Data to AI Foundation and Sending Data to Planning
Batch Processing	Enable daily and weekly batch processing and establish all ongoing integrations with Oracle and non-Oracle systems.	Complete the chapter on Batch Orchestration
Cutovers	Perform mock cutovers between Pre-Production and Production environments, where data and batches are tested in Production using final batch flows.	Environment cutovers are scheduled through My Oracle Support
	Plan for final cutover to production several weeks before the go-live date and establish an outage window based on the duration of the mock cutover.	Environment cutovers are scheduled through My Oracle Support

Based on all of the topics listed above, here is an example of some major milestones and key activities that might be followed for a project that includes Merchandise Financial Planning or IPOCS-Demand Forecasting:

1. Oracle sends you a welcome mail having the required credentials to access your Retail Analytics and Planning applications - RI, AI Foundation, MFP, IPO, AP, POM, RH, DV, Apex, Innovation Workbench. You will receive a combined email pointing you to Retail Home for the individual application links.

2. Verify that the modules purchased by the customer are enabled in Retail Home during deployment. Review the steps in the *RAP Implementation Guide* on managing customer modules.
3. Verify that you can access POM and that the batch schedules for your subscribed applications are available.
4. Prepare the scripts to access object storage and test your connection to the File Transfer Services (FTS).
5. Apply initial configurations to all your applications per the documentation for each solution.
6. Upload the files required by RAP for foundation and historical data to object storage.
7. Run the first set of ad hoc jobs to load data into RI's interfaces, following the *RAP Implementation Guide* and *RAP Operations Guides* as needed.
8. Run the next set of ad hoc jobs to publish data to AI Foundation and Planning (PDS), following the *RAP Implementation Guide* and *RAP Operations Guides* as needed.
9. Repeat the ad hoc data load process iteratively until all history is loaded, and perform any needed seeding steps per the *RAP Implementation Guide* to establish full snapshots of positional data.
10. Run the PDS domain build activity to build your Planning domain (this can be done as soon as you have any data moved to PDS; it does not require completing the history load).
11. Create user groups in MFP/IPO/AP and configure access in OCI IAM for business users.
12. Configure forecast settings in AI Foundation for generating forecasts and validate forecast execution works as expected.
13. Upload files to object storage for your complete nightly batch runs and initiate the full batches for RAP.

2

Setup and Configuration

The Setup and Configuration chapter provides parameters and steps for setting up a new Retail Analytics and Planning cloud environment. While the platform comprises many application modules (some of which you may not use), there are certain common processes and settings that are shared across all of them. It is critical to check and update these core settings before moving on to later implementation steps, as they will define many system-wide behaviors that could be difficult to change once you've started loading data into the platform.

Configuration Overview

A high-level outline of the setup process is provided below to describe the activities to be performed in this chapter.

Table 2-1 RAP Configuration Overview

Activity	Description
Learn the configuration tools	The Retail Analytics and Planning has many tools available to support an implementation, such as Retail Home, POM, and APEX. Knowing how to use these tools is an important first step in the process. Review the Implementation Tools chapter for details.
Verify Object Storage connectivity	Generate access tokens for interacting with Object Storage and test the connection, as it is required for all file movement into and out of the Oracle cloud. Review the Implementation Tools chapter for details.
Configure the system calendar	Update the parameters that define the type and characteristics of your business calendar, such as the start and end dates RAP will use to define calendar generation.
Configure the system languages	Update the master list of supported languages that need to be present in addition to your primary language, such as the need for seeing data in both English and French.
Configure history retention policies	Certain data tables in Retail Insights that are leveraged by other applications on the platform have a history retention period after which some data may be erased.
Configure application-specific settings	All applications in the Retail Analytics and Planning have their own settings which must be reviewed before starting an implementation of those modules.

Platform Configurations

This section provides a list of initial setup and configuration steps to be taken as soon as you are ready to start a new implementation of the Retail Analytics and Planning and have the cloud environments provisioned and generally available.

Several configuration tables in the RAP database should be reviewed before processing any data. A list of these tables is below, along with an explanation of their primary uses. The way to apply changes to these tables is through the Control & Tactical Center, as described in the section on Control & Tactical Center. The sections following this one provide the detailed configuration settings for each table listed below.

Table 2-2 Platform Configuration Table Overview

Table	Usage
C_ODI_PARAM (C_ODI_PARAM_VW)	Table used to configure all Oracle Data Integrator (ODI) batch programs as well as many Retail Insights and AI Foundation load properties. `C_ODI_PARAM_VW` is the name of the table shown in Control Center.
W_LANGUAGES_G	Table used to define all languages that need to be supported in the database for translatable data (primarily for Retail Insights and AI Foundation Cloud Services).
C_MODULE_ARTIFACT	Table used for database table partitioning setup. Defines which functional areas within the data warehouse will be populated (and thus require partitioning).
C_MODULE_EXACT_TABLE	Table used to configure partition strategies for certain tables in the data warehouse, including the Plan fact used for loading plans and budgets to RI/AI Foundation.
C_MODULE_DBSTATS	Table used to configure the ad hoc manual stats collection program `COLLECT_STATS_JOB`.
C_HIST_LOAD_STATUS	Table used to configure historical data loads, configure certain ad hoc batch processes, and monitor the results of those jobs after each run.
C_HIST_FILES_LOAD_STATUS	Table used to track multiple zip files that are uploaded with sequence numbers in order to process them automatically through ad hoc flows.
C_SOURCE_CDC	Table used to configure and monitor both historical and ongoing integration to Planning applications through the Retail Insights data warehouse.
C_DML_AUDIT_LOG	Audit table used to track updates to the `C_ODI_PARAM` table by users from APEX or Control Center.

C_ODI_PARAM Initialization

The first table requiring updates is `C_ODI_PARAM` because your system calendar is populated using the ODI programs. This table is displayed as `C_ODI_PARAM_VW` on the **Manage System Configurations** screen in the Control & Tactical Center. The following settings must be updated prior to using the platform. These settings are required even if your project only includes Planning implementations. Changes to these settings are tracked using the audit table `C_DML_AUDIT_LOG`.

Table 2-3 C_ODI_PARAM Initial Settings

Scenario Name	Param Name	Configuration Guidance
SIL_DAYDIMENSION	START_DT	Start date for generating the Gregorian calendar (this is different from the fiscal calendar). Set 12+ months before the start of the planned fiscal calendar to provide adequate space for adjustments to the fiscal calendar starting period. Do not set `START_DT` to be in the middle of a fiscal year that is in your calendar file. If you pick a `START_DT` that is later than the earliest period in your file, then the `START_DT` must fall on day 1 of a fiscal year, or the data will be incorrect when loaded. Example: If your first Fiscal start date is in February 2020, then it would be fine to start the Gregorian calendar on `20190101`. Starting from the first day of a year ensures there are no incomplete months in the Gregorian calendar. Note that `START_DT` is well before the start of the Fiscal Calendar. If you are loading the calendar directly from Merchandising, refer to section Using RDE for Calendar Setup (Gen 2 Architecture).
SIL_DAYDIMENSION	END_DT	End date for generating the Gregorian calendar (this is different from the fiscal calendar), non-inclusive. Set 6-12 months beyond the expected end of the fiscal calendar to ensure the final year of that calendar does not extend beyond the available dates. The `END_DT` will be automatically updated based on subsequent loads of `CALENDAR.csv` to ensure the `END_DT` is never earlier than the last date in the file. Example: If your Fiscal end period is currently January 2025, then you could set the Gregorian `END_DT` to `20260101`. This will end the system calendar on `20251231` (December 31, 2025), because the `END_DT` itself is not used. Ending on the last day of a year ensures there are no incomplete months in the calendar.
SIL_DAYDIMENSION	WEEK_START_DT_VAL	Starting day of the week for both Gregorian and Fiscal calendars (1 = Sunday, 2 = Monday, and so on through 7 = Saturday). The default calendar setup uses a Sunday-to-Saturday week.
GLOBAL	START_OF_YEAR_MONTH	The name of the Gregorian month associated with the first fiscal period in your business calendar. For example, if your fiscal year starts 06-FEB-22 then set this to `FEBRUARY`. This will be used to display month names in RI reporting on the fiscal calendar. Default = `JANUARY`

Table 2-3 (Cont.) C_ODI_PARAM Initial Settings

Scenario Name	Param Name	Configuration Guidance
GLOBAL	RI_OPTIONALLY_ENCLOSED_BY	**Note:** This parameter is deprecated in the new RAP architecture and was replaced by CTX file parameters. Set a character to use for wrapping text strings in data files, such as a quotation mark ("), to allow column delimiters to occur within the strings without causing any failures in the load process. The recommended value is "
GLOBAL	CURRENCY_CODE	Set the default currency code used when loading CSV-based fact data files if none are provided on the files themselves. Defaults to 'USD'.
GLOBAL	HIST_ZIP_FILE	Change the default name for the ZIP file package used by the history file load process. Default=`RAP_DATA_HIST.zip`
GLOBAL	LANGUAGE_CODE	Default language code used by the system to load data. Do not change unless your source systems are using a non-English primary language in their database and datasets. Default=EN
GLOBAL	RI_INV_HIST_DAYS	The number of days to retain a zero-balance record on inventory positions. Excessive retention of zero balances can cause batch performance issues due to high data volumes. But dropping the records too soon may be detrimental to your business reporting or analytical processes if you make use of zero-balance information. Default=91 days.
GLOBAL	RA_INV_WAC_IND	Controls the RDE inventory cost calculation. When set to `Y`, it will use Weighted Average Cost (WAC) as the item cost for all items. When set to `N`, it will dynamically load Merchandising valuation methods set per department or item and apply them, choosing from average cost, unit cost, or retail-based cost.
GLOBAL	RA_INV_TAX_IND	Controls the RDE retail calculation and removal of tax amounts from retail valuation of stock on hand and on-order amounts. When set to `N`, only simple VAT (SVAT) calculations are supported and taxes are included in the values. When set to `Y`, the system dynamically loads Merchandising global tax and VAT information and applies it by item/location to remove taxes.

Table 2-3 (Cont.) C_ODI_PARAM Initial Settings

Scenario Name	Param Name	Configuration Guidance
GLOBAL	RA_SLS_TAX_IND	Controls the RDE retail calculation and removal of tax amounts from retail valuation of sales amounts. When set to `N`, it is generally VAT-inclusive on the Retail amounts (but not profit amounts, which never include VAT). When set to `Y`, the system dynamically loads Merchandising VAT information and applies it by item/location to remove VAT taxes from all sales retail and discount amounts.
GLOBAL	RI_CLOSED_PO_HIST_DAYS	The number of days to retain closed purchase orders on the daily positional snapshots. Closed purchase orders may be important for reporting or analytical processes, but typically are not needed as they do not impact your open on-order calculations. Default=30 days.
GLOBAL	RI_GEN_PROD_RECLASS_IND	Set to `Y` to enable AIF foundation loads to automatically generate item-level reclass records from your hierarchy data. From version 24 onwards, this is the default method to handle reclasses. Requires that the product file in the nightly batch include any reclass changes so the system can detect when an item moves between hierarchy positions even if no other change occurred. Default = `Y`.
GLOBAL	RI_INT_ORG_DS_MANDATORY_IND	Set to `Y` to require input data on the Organization hierarchy interface for the batch to run. This will prevent the batch from executing if the data files were not uploaded properly for a given day or the file was missing from the upload.
GLOBAL	RI_PROD_DS_MANDATORY_IND	Set to `Y` to require input data on the Product hierarchy interface for the batch to run. This will prevent the batch from executing if the data files were not uploaded properly for a given day or the file was missing from the upload.
GLOBAL	RI_LAST_MKDN_HIST_IND	Set to `Y` to enable the Price fact columns for LPO (`LST_MKDN_RTL_AMT_LCL`, `LST_MKDN_DT`, `LST_PROMO_RTL_AMT_LCL`, `LST_PROMO_DT`) to be populated during history loads. This will impact performance of the loads and is disabled by default.

Table 2-3 (Cont.) C_ODI_PARAM Initial Settings

Scenario Name	Param Name	Configuration Guidance
GLOBAL	RI_ITEM_REUSE_IND	Enable or disable the ability to re-use item numbers over time to represent entirely new items. Also enables retention of existing items for a number of days, so that if an item drops and reappears quickly, it is not considered a new item and will continue to use the existing records. Set to `Y` to enable. If this is not enabled, items that are dropped from the product interface are immediately closed and deactivated and cannot be re-opened. Default = N
GLOBAL	RI_ITEM_REUSE_AFTER_DAYS	The number of days between when an item is deleted and when it's allowed to appear as a new item having the same ID. This will trigger the old version of the item to be archived in the data warehouse using an alternate key, so the new version of the item is treated as completely new. For example, setting this to 5 days means that an item can be dropped/ deleted and after 5 days, when the same items comes again it will be treated as a brand new item. If the item re-appears in the data before 5 days has passed, it will be treated as the same item as before and the existing item data remains active. Default = 0
GLOBAL	RI_LOC_CLOSE_AFTER_DAYS	The number of days a location record will be kept active after it has been dropped from the inbound interfaces such as `ORGANIZATION.csv`. After the number of days is passed, the location will be closed, marked inactive, and may not be re-added to the file again. Default = 0
GLOBAL	ITEM_PARENT_DIFF_SEPARATOR	Set the field separator used when constructing the `ITEM_PARENT_DIFF` level of the product hierarchy export to PDS. This character will be used to concatenate the item ID and differentiator ID. Default = _
GLOBAL	PDS_PROD_INCLUDE_ITEM_ID	Control whether item identifiers are included in the PDS product labels or not. When set to a value of `N`, only the product descriptions are included in the labels. When changed to a value of `Y`, the item IDs are concatenated in front of the descriptions on `W_PDS_PRODUCT_D`.

Table 2-3 (Cont.) C_ODI_PARAM Initial Settings

Scenario Name	Param Name	Configuration Guidance
GLOBAL	PDS_EXPORT_DAILY_ONORD	Determines whether the `EOW_DATE` used in Purchase Order export data is allowed to contain non-end-of-week (EOW) dates, or if the system must convert it to a week-ending date in all cases. When set to a value of `Y`, daily dates are allowed in the `EOW_DATE` field on an export (if there is a daily date in the `OTB_EOW_DATE` column of `ORDER_HEAD.csv`). When set to a value of `N`, it means the system automatically converts the input dates from `ORDER_HEAD.csv` to week-ending dates only.
SIL_RETAILINVPOSITIONFACT	INV_FULL_LOAD_IND	Control the Inventory Position fact load behavior. When set to `N` it uses the incremental update behavior where it requires just the changes to inventory to be posted, including zeros. When set to `Y`, it assumes full nightly snapshots are being loaded and automatically zeroes out all inventory positions not sent for that load. Default = `N`.
SIL_RETAILPOONORDERFACT	PO_FULL_LOAD_IND	Control the Purchase Order fact load behavior. When set to `N` it uses the incremental update behavior where it requires just the changes to POs the be posted, including zeros. When set to `Y`, it assumes full nightly snapshots are being loaded and automatically zeroes out all POs not sent for that load. Default = `N`.
SIL_RETAIL_COHEADDIMENSION	RI_MIS_COHEAD_REQ_IND	Seed missing customer order (CO) head IDs from sales fact to CO Dimension. If you are providing customer order IDs on your sales history load, make sure to set this to `Y`.
SIL_RETAILCOLINEDIMENSION	RI_MIS_COLINE_REQ_IND	Seed missing customer order (CO) line IDs from sales fact to CO Dimension. If you are providing customer order line IDs on your sales history load, make sure to set this to `Y`.
SIL_EMPLOYEEDIMENSION	RI_MIS_CASHIER_REQ_IND	Seed missing Cashier IDs from sales fact to Employee dimension. If you are providing employee IDs on your sales history load, make sure to set this to `Y`.
SIL_RETAILCUSTOMERDIMENSION	RI_MIS_CUSTOMER_REQ_IND	Seed missing customer IDs from sales fact to Customer dimension. If you are providing customer IDs on your sales history load, make sure to set this to `Y`.
SIL_RETAILPROMODIMENSION	RI_MIS_PROMO_REQ_IND	Seed missing promotions from the sales promo fact to the Promotion dimension. If you are providing promotion IDs on your sales history load and not providing a Promotion file, make sure to set this to `Y`.

The following key decisions must be made during this initial configuration phase and the proper flags updated in C_ODI_PARAM:

- **Item Number Re-Use** – If you expect the same item numbers to be re-used over time to represent new items, then you must update `RI_ITEM_REUSE_IND` to `Y` and `RI_ITEM_REUSE_AF TER_DAYS` to a value >=1. Even if you are not sure how item re-use will occur, it's better to enable these initially and change them later as needed.
- **Tax Handling** – Both for historical and ongoing data, you must decide how tax will be handled in fact data (will tax amounts be included or excluded in retail values, what kind of tax calculations may be applied when extracting history data, and so on). You may or may not need any configurations updated depending on your RDE usage.
- **Full vs Incremental Positional Loads** – In nightly batches, the core positional fact loads (Purchase Orders and Inventory Positions) support two methods of loading data: full snapshots and incremental updates. You must decide which of these methods you will use and set `INV_FULL_LOAD_IND` and `PO_FULL_LOAD_IND` accordingly. Incremental updates are preferred, as they result in lower data volumes and faster nightly batch performance; but not all source systems support incremental extracts.

If you are using RDE to integrate with Merchandising, pay special attention to the global tax and WAC configurations, as these control complex calculations that will change how your data comes into RAP. These options should not be changed once you enable the integrations because of the impact to the daily data. For example, a large European retailer with presence in multiple VAT countries may want the following options:

- `RA_INV_WAC_IND = N` - This will dynamically calculate inventory cost using all three Merchandising cost methods instead of just using WAC
- `RA_INV_TAX_IND = Y` - This will enable the removal of tax amounts from retail values so inventory and PO reporting is VAT-exclusive
- `RA_SLS_TAX_IND = Y` - This will enable the removal of tax amounts from retail values so sales reporting is VAT-exclusive

Retail Insights contains many additional configurations in the `C_ODI_PARAM` table that are not necessary for platform initialization, but may be needed for your project. This includes Merchandise Financial Planning and IPOCS-Demand Forecasting configurations for specifying custom planning levels to be used in the integration between MFP/IPO and RI (when RI will be used for reporting). The default parameters align with MFP/IPO default plan outputs, but if you are customizing them to use a different base intersection, then you must also update those values in `C_ODI_PARAM`. Refer to the *Retail Insights Implementation Guide* for complete details on Planning Configurations.

W_LANGUAGES_G Initialization

The `W_LANGUAGES_G` table controls all the languages supported in the translatable database data. This applies to areas such as product names, location names, attribute values, season/ phase descriptions, and other text-based descriptors. Additional languages are used mainly by Retail Insights, which supports displaying data in multiple languages in reporting and analytics. It is required to delete all languages from this table that will not be used because every language code in this table will have records generated for it in some interfaces, creating unnecessary data that can impact system performance. Starting with version 23.1.201.0, new environments will only be created with the 'US' language code in place; but if you are on an earlier version then you must manually delete all other entries that will not be used.

For example, product names will automatically have database records initialized for every supported language in this configuration table, even if the data you are providing does not contain any of those languages. This creates significant amounts of data in your product descriptions table, which may not serve any real purpose for your implementation. If you are only using a single primary language, then you can safely delete all but one row from `W_LANGUAGES_G`. The default row to preserve is the one with a language code of `US` which is used for American English.

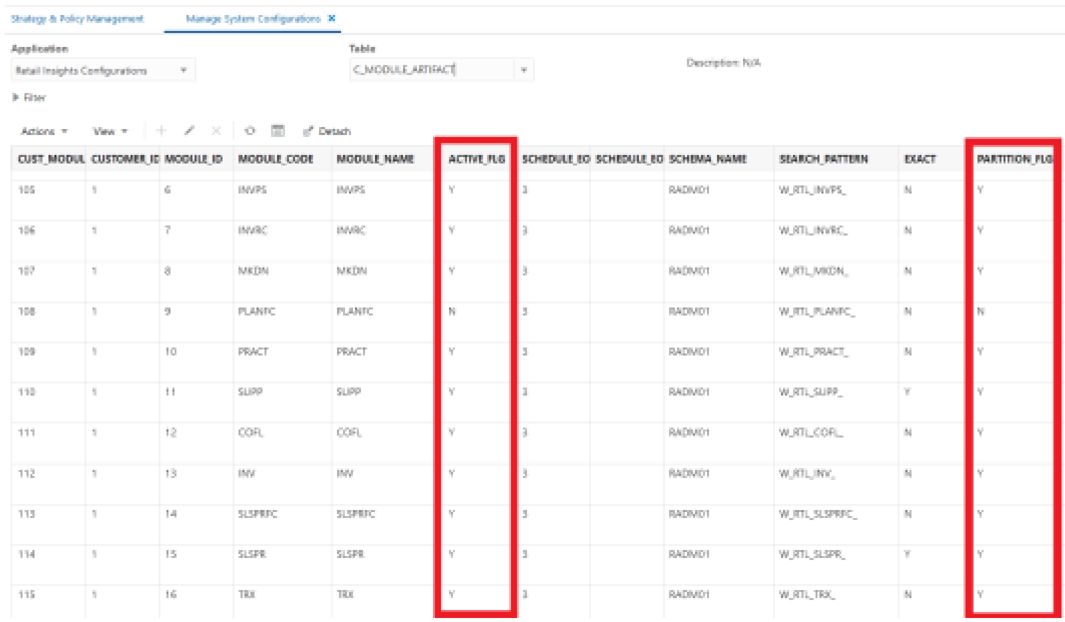
C_MODULE_ARTIFACT Initialization

> **Note:**
>
> In version 25, manual partitioning steps have been deprecated and replaced by automated partitioning. The automated partitioning process does not rely on the `ACTIVE_FLG` or `PARTITION_FLG` values set below, it will perform partitioning even if the flag value is `N`. However, manual partition jobs do rely on these flags, so it is still recommended to review these configurations at the start of a new implementation project.

The `C_MODULE_ARTIFACT` table is used by the database to configure table partitioning. Many tables in the platform are partitioned based on the business calendar (usually by calendar date or fiscal week) and require this setup step before you start loading data. You should perform this step regardless of which application modules you are implementing, because all foundation data passes through this architecture. Partitions will be automatically generated at runtime. For example, when you load sales transaction history, the fact load will create partitions to hold the incoming data based on the configurations in this table.

Before you run any jobs to load data into RAP, you must validate this table has all rows set to `ACTIVE_FLG=Y` and `PARTITION_FLG=Y` with the exception of the SLSPRFC module, which should not be partitioned at this time and must have flag values of `N` instead. Additionally, you must validate the date set on column `PARTITION_HISTORY_DATE` is sufficient for your planned range of history data. By default, the column is set to `2021-12-23`, meaning that partitions will only be created from this date forwards. If you plan to load history data for periods earlier than this, then update the value of `PARTITION_HISTORY_DATE` on all rows to be earlier than your first day of history.

You also must choose whether you are planning to load the Planning facts (such as `W_RTL_PLAN1_PROD1_LC1_T1_FS`) for plan/budget data in RI or AI Foundation. If you are not using the table right away, you should also disable the PLAN modules, like PLAN1. You can revisit this setup later to perform additional partitioning as needed.



CUST_MODUL	CUSTOMER_ID	MODULE_ID	MODULE_CODE	MODULE_NAME	ACTIVE_FLG	SCHEDULE_EO	SCHEDULE_EO	SCHEMA_NAME	SEARCH_PATTERN	EXACT	PARTITION_FLG
105	1	6	INVPS	INVPS	Y	3		RADM01	W_RTL_INVPS_	N	Y
106	1	7	INVRC	INVRC	Y	3		RADM01	W_RTL_INVRC_	N	Y
107	1	8	MKDN	MKDN	Y	3		RADM01	W_RTL_MKDN_	N	Y
108	1	9	PLANFC	PLANFC	N	3		RADM01	W_RTL_PLANFC_	N	N
109	1	10	PRACT	PRACT	Y	3		RADM01	W_RTL_PRACT_	N	Y
110	1	11	SLIPP	SLIPP	Y	3		RADM01	W_RTL_SLIPP_	Y	Y
111	1	12	COFL	COFL	Y	3		RADM01	W_RTL_COFL_	N	Y
112	1	13	INV	INV	Y	3		RADM01	W_RTL_INV_	N	Y
113	1	14	SLSPRFC	SLSPRFC	Y	3		RADM01	W_RTL_SLSPRFC_	N	Y
114	1	15	SLSPR	SLSPR	Y	3		RADM01	W_RTL_SLSPR_	Y	Y
115	1	16	TRX	TRX	Y	3		RADM01	W_RTL_TRX_	N	Y

C_MODULE_EXACT_TABLE Initialization

The `C_MODULE_EXACT_TABLE` table is used for defining flexible partitioning strategies on certain tables. Most data in this table can be left as-is, but you must update this table if you plan to load Planning or Budget information into the `W_RTL_PLAN1_PROD1_LC1_T1_FS` interface. The partition level must align with the data level of your plan (day or week). To configure the plan partitions, you must update the table `C_MODULE_EXACT_TABLE` where `MODULE_CODE = PLAN1`. Modify the columns `PARTITION_COLUMN_TYPE` and `PARTITION_INTERVAL` to be one of the following values:

- If your input data will be at Day level, set both columns to `DY`
- If your input data will be at Week level, set both columns to `WK`

You must then enable the partitioning process in `C_MODULE_ARTIFACT` by locating the row for `MODULE_CODE=PLAN1` and setting `ACTIVE_FLG=Y` and `PARTITION_FLG=Y`. If your plan data will extend into the future, you must also change `PARTITION_FUTURE_PERIOD` to the number of future months that need partitions built (for example, use a value of `6M` to partition 6 months into the future).

C_HIST_LOAD_STATUS

The `C_HIST_LOAD_STATUS` table is used to track the progress of historical loads of data, primarily inventory position and pricing facts. You should edit the following fields on this table based on your implementation needs:

- `HIST_LOAD_LAST_DATE` – Specifies the planned final date for the end of your historical loads (for example, the end of the 2-year period you plan to load into RAP). The history load programs will assume that you are providing each week of inventory in sequence from earliest to latest and process the data in that order.
- `ENABLED_IND` – Turns on or off a specific table load for historical data. Most of the tables in these processes are only required for Retail Insights, and the rest can be disabled to improve performance. Set to a value of `N` to disable a table load.

- MAX_COMPLETED_DATE – The load programs use this to keep track of the last loaded week of data. It does not allow you to reload this week or any prior week, so if you are trying to start over again after purging some history, you must also reset this field.
- HIST_LOAD_STATUS – The load programs uses this to track the status of each step in the load process. If your program gets stuck on invalid records change this field back to INPROGRESS before re-running the job. If you are restarting a load after erasing history data, then you need to clear this field of any values.

If you are implementing Retail Insights, then enable all INV and PRICE modules in the table (set ENABLED_IND to Y). If you are only implementing AI Foundation or Planning application modules, then the following history tables should be enabled; all others should be disabled (set ENABLED_IND to N).

- W_RTL_PRICE_IT_LC_DY_F
- W_RTL_PRICE_IT_LC_DY_HIST_TMP
- W_RTL_INV_IT_LC_DY_F
- W_RTL_INV_IT_LC_WK_A
- W_RTL_INV_IT_LC_DY_HIST_TMP

After enabling your desired history load tables, update the value of HIST_LOAD_LAST_DATE on all rows you enabled. Set the date equal to the final date of history to be loaded. This can be changed later if you need to set the date further out into the future.

As you load data files for one or more weeks of history per run, the value of MAX_COMPLETED_DATE and HIST_LOAD_STATUS automatically update to reflect the progress you have made. If you need to restart the process (for example, you have loaded test data and need to start over with production data) these two columns must first be cleared of all data from the Control Center before beginning the history load again.

C_SOURCE_CDC

The C_SOURCE_CDC table is used for changed data capture (CDC) parameters for the integrations between the Retail Insights data warehouse and the Planning application schemas. In general, this table is updated automatically as batches are run. However, it is important to know when you may need to modify these values.

For most interfaces, the table will initially have no records. The first time an integration batch program runs, it will take all the data from the source table and move it to the export table. It will then create a C_SOURCE_CDC record for the target table name, with a value for LAST_MIN_DATE and LAST_MAX_DATE matching the timeframe extracted. On the next run, it will look at LAST_MAX_DATE as the new minimum extract date and pulls data greater than that date from the source table. If you are performing history loads for tables, such as Sales Transactions, you may need to change these dates if you have to re-send data to Planning for past periods.

Specifically for positional data (at this time only Inventory Position), the usage is not quite the same. Positional data will always send the current end-of-week values to Planning, it does not look at historical weeks as part of the normal batch process. A separate historical inventory integration program is provided in an ad hoc process, which will allow you to send a range of weeks where LAST_MIN_DATE is the start of the history you wish to send, and LAST_MAX_DATE is the final date of history before normal batches take it forward. It is common to load inventory from end to end in isolation as it is a data-intensive and time-consuming process to gather, load, and validate inventory positions for multiple years of history.

W_GLOBAL_CURR_G

The W_GLOBAL_CURR_G table is used by Retail Insights to support up to three additional currencies in reporting and aggregation (other fields above 3 are not used at this time). RI pre-populates global currency fields in all aggregation tables based on the specified currency codes. The desired codes are added to one row in this table and must align with the Exchange Rates data provided separately. This table is available from the Control & Tactical Center and is not a required configuration for any project unless you wish to report on additional currencies in Retail Insights.

Example data to be inserted to this table:

DATASOURCE_NUM_ID	TENANT_ID	GLOBAL1_CURR_CODE	GLOBAL2_CURR_CODE	GLOBAL3_CURR_CODE	GLOBAL1_RATE_TYPE	GLOBAL2_RATE_TYPE	GLOBAL3_RATE_TYPE	DEFAULT_LOC_RATE_TYPE
1	DEFAULT	INR	AED	PEN	Corporate	Corporate	Corporate	Corporate

Application Configurations

In addition to the platform configurations defined above, each application on the platform has its own system and runtime options that need to be reviewed and updated. The information below will guide you to the appropriate content for each application's configuration options.

Retail Insights

Retail Insights has a significant number of configurations, primarily in the `C_ODI_PARAM_VW` table in the Control Center, which controls batch processes and reporting behaviors throughout the application. If you are implementing Retail Insights as part of your project, review the "Setup and Configuration" chapter of the *Retail Insights Implementation Guide*.

AI Foundation Cloud Services and Forecasting

Each AI Foundation application has parameters that are specific to the batch processing, data movement, algorithms, and user interfaces of those modules. These configurations are stored in several database tables available through the Control & Tactical Center. If you are implementing any AI Foundation applications as part of your project, review the *Retail AI Foundation Cloud Services Implementation Guide*.

If you are implementing any planning application (Merchandise Financial Planning, IPOCS-Demand Forecasting, or Assortment Planning), then you are required to configure and use the Forecasting module in the AI Foundation application interface. This requires initial configurations to select forecast parameters, as well as post-data load configurations to select forecast data levels and perform testing of the chosen algorithm. For basic information about Forecasting and what the AI Foundation application functionality can support, refer to the "Manage Forecast Configurations" section in the *AI Foundation User Guide*.

To configure the forecast process for Planning, use the **Manage System Configurations** screen in the Control Center to review and modify the configurations in RSE_CONFIG. These values can be set up now, but you cannot complete the rest of the forecasting process until your foundation data has been loaded into AI Foundation.

Appl Code	Parameter Name	Description
RSE	LOAD_EXTENDED_PROD_HIER	Y or N value (default value is Y). Extended hierarchy refers to a 9-level structure with style and style/color as extra levels above SKU. This is used only for specific applications such as AP, IPO, and LPO. If you are not using one of the listed applications or you don't have styles and style/colors, then you can ignore this parameter (the extended hierarchy won't be used even if generated).
RSE	EXTENDED_HIERARCHY_SRC	Value can be set as either RMS or NON-RMS. Default value is NON-RMS. If using RMFCS or RMS-sourced data, or you are loading RAP with data in RMS-like format, change this value to RMS. Data loaded using the RAP foundation CSV files can be provided in either an RMS or non-RMS format, but RMS format is preferred (as detailed later in this document for the PRODUCT.csv file). All interface samples use RMS-formatted data, so change this parameter to RMS if you are following those guidelines.
PMO	PMO_PROD_HIER_TYPE	The hierarchy ID to use for the Lifecycle Pricing Optimization product and the Forecasting module. AIF applications have 2 product hierarchies: • 1 = Basic 7-level hierarchy without styles or colors • 3 = Extended 9-level hierarchy with style/color Default value is 3. If the extended hierarchy is enabled and you are using one of the apps listed above on LOAD_EXTENDED_PROD_HIER, keep this value as 3. If you are not using the extended hierarchy (such as for MFP-only implementations), change this value to 1.
RSE	PROD_HIER_SLSTXN_HIER_LEVEL_ID	This parameter identifies the extended hierarchy level at which sales transactions are provided (7-Style, 8-Style/color or 9-Style/color/Size). It MUST match the extended hierarchy leaf level. Default value is 9. If you are not using the extended hierarchy then ignore this parameter, it will not be used.
PMO	PMO_AGGR_INVENTORY_DATA_FLG	Specifies whether inventory data is present and if it should be used when aggregating activities data for LPO and forecasting (only some forecast types use inventory). Set this value to N if inventory data is not loaded or not needed for forecasting (inventory data is not used for MFP forecasting but it is required for other applications like Lifecycle Pricing Optimization and Inventory Planning Optimization). Default value is Y.

Planning Platform

Planning Applications such as MFP (Merchandise Financial Planning) can be set up using the Planning Platform (RPASCE). It allows customers to use a Standard GA template version or configurable planning solution versions. Refer to the Planning application-specific Implementation Guides for more details about these options.

3

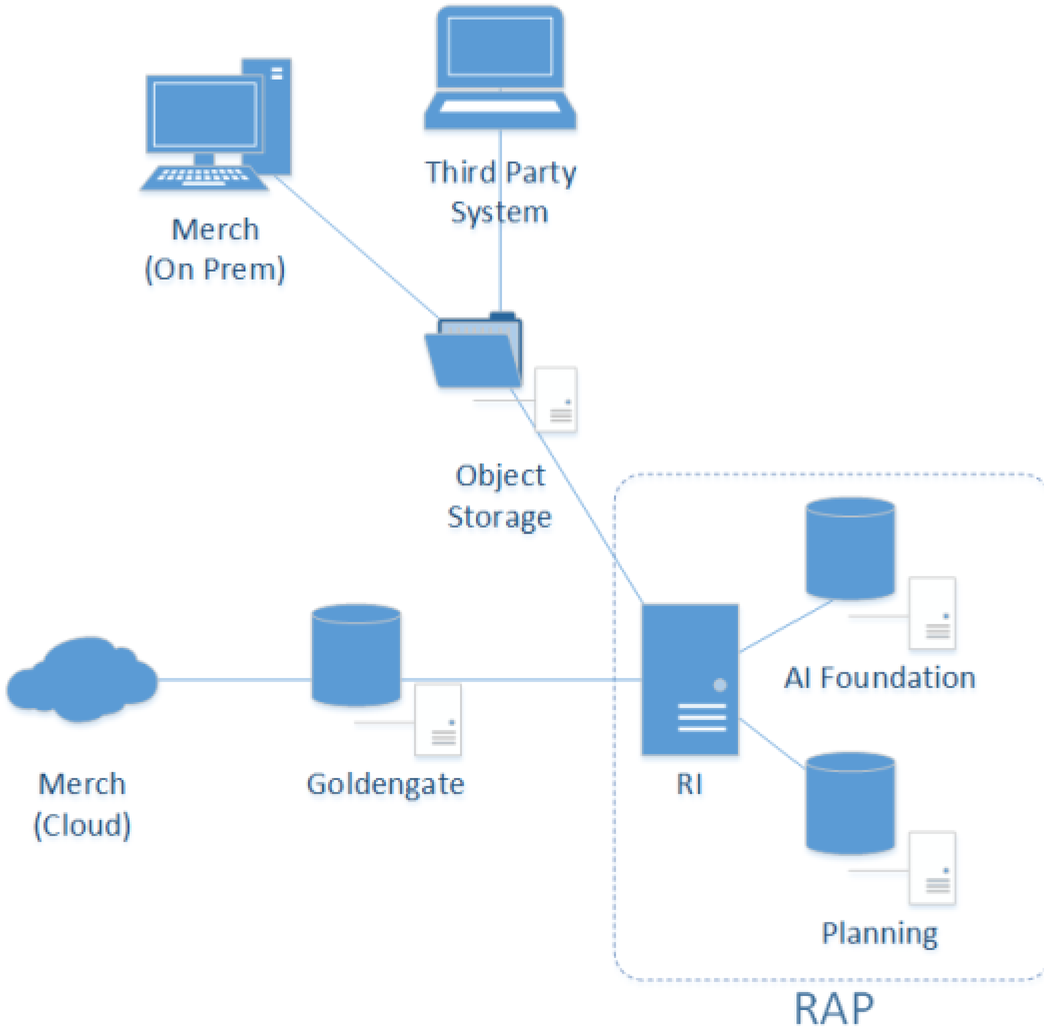
Data Loads and Initial Batch Processing

This chapter describes the common data requirements for implementing any of the Retail Analytics and Planning modules, where to get additional information for optional or application-specific data interfaces, and how to load an initial dataset into the cloud environments and distribute it across your desired applications.

Data Requirements

Preparing data for one or more Retail Analytics and Planning modules can consume a significant amount of project time, so it is crucial to identify the minimum data requirements for the platform first, followed by additional requirements that are specific to your implementation plan. Data requirements that are called out for the platform are typically shared across all modules, meaning you only need to provide the inputs once to leverage them everywhere. This is the case for foundational data elements, such as your product and location hierarchies. Foundation data must be provided for any module of the platform to be implemented. Foundation data is provided using different sources depending on your current software landscape, including the on-premise Oracle Retail Merchandising System (RMS) or 3rd-party applications.

Figure 3-1 Inbound Foundation Data Flows



Retail Insights is used as the foundational data warehouse that collects and coordinates data on the platform. You do not need to purchase Retail Insights Cloud Service to leverage the data warehouse for storage and integration; it is included as part of any RAP solution. Regardless of which RAP solutions you are implementing, the integration flows shown above are used.

Application-specific data requirements are in addition to the shared foundation data, and may only be used by one particular module of the platform. These application data requirements may have different formats and data structures from the core platform-level dataset, so pay close attention to those additional interface specifications. References and links are provided later in this chapter to guide you to the relevant materials for application-specific inputs and data files.

If you are using RMS as your primary data source, then you may not need to produce some or all of these foundation files, as they will be created by other Oracle Retail processes for you. However, it is often the case that historical data requires a different set of foundation files from your future post-implementation needs. If you are loading manually-generated history files, or you are not using an Oracle Retail data source for foundation data, then review the rest of this section for details.

Platform Data Requirements

There is a subset of core platform data files that can be created and loaded once and then used across some or all application modules. These files use a specific format as detailed below.

> **Note:**
>
> Every application included with Retail Analytics and Planning has additional data needs beyond this foundation data. But this common set of files can be used to initialize the system before moving on to those specific requirements.

The first table defines the minimum dimensional data. A dimension is a collection of descriptive elements, attributes, or hierarchical structures that provide context to your business data. Dimensions tell the platform what your business looks like and how it operates. This is not the entire list of possible dimension files, just the main ones needed to use the platform. Refer to Legacy Foundation File Reference for a complete list of available platform foundation files, along with a cross-reference to the legacy interfaces they most closely align with. A complete interface specification document is also available in My Oracle Support to assist you in planning your application-specific interface needs.

Table 3-1 Common Foundation Dimensions

Dimension	Filename(s)	Usage
Product	PRODUCT.csv	The product foundation data includes the items you sell, their core merchandise attributes, and their hierarchical structure in your source systems.
Product	PRODUCT_ALT.csv	Alternate product attributes and hierarchy levels intended for downstream Planning application extensions.
Organization	ORGANIZATION.csv	The organization foundation data includes all of your business entities involved in the movement or sale of merchandise. This includes stores, warehouses, partner/finisher locations, web stores, and virtual warehouses. It also includes your organizational hierarchy and core location attributes.
Organization	ORGANIZATION_ALT.csv	Alternate location attributes and hierarchy levels intended for downstream Planning application extensions.
Calendar	CALENDAR.csv	The calendar foundation data defines your business (or fiscal) calendar. This is the calendar that you operate in when making critical business decisions, reporting on financial results, and planning for the future. The most common calendar used by all modules of the platform is a 4-5-4 fiscal calendar.

Table 3-1 (Cont.) Common Foundation Dimensions

Dimension	Filename(s)	Usage
Exchange Rates	EXCH_RATE.csv	Exchange rates define the conversion of monetary values from the currency they are recorded in to your primary operating currency. Most data is provided to the platform in the local currency of the data source, and it is converted to your primary currency during the load process.
Product Attributes	ATTR.csv PROD_ATTR.csv	Product attributes describe the physical and operational characteristics of your merchandise and are a critical piece of information for many AI Foundation modules, such as Demand Transference and Size Profile Optimization. They are not required as an initial input to start data loads but will eventually be needed for most applications to function.
System Parameters	RA_SRC_CURR_PARAM_G.dat	Parameters file that supports certain batch processes, such as the ability to load multiple ZIP files and run batches in sequence. Include this file with nightly batch uploads to specify the current business date, which enables the system to run multiple batches in sequence without customer input. Required once you begin nightly or weekly batch uploads.

The other set of foundation files are referred to as facts. Fact data covers all of the actual events, transactions, and activities occurring throughout the day in your business. Each module in the platform has specific fact data needs, but the most common ones are listed below. At a minimum, you should expect to provide **Sales**, **Inventory**, and **Receipts** data for use in most platform modules. The intersection of all data (meaning which dimensional values are used) is at a common level of **item/location/date**. Additional identifiers may be needed on some files; for example, the sales data should be at the transaction level, the inventory file has a clearance indicator, and the adjustments file has type codes and reason codes.

Table 3-2 Common Foundation Facts

Dimension	Filename(s)	Usage
Sales	SALES.csv	Transaction-level records for customer sales (wholesale data provided separately). Used across all modules.
Inventory	INVENTORY.csv	Physical inventory levels for owned merchandise as well as consignment and concession items. Used across all modules.
Receipts	RECEIPT.csv	Inventory receipts into any location, including purchase order and transfer receipts. Used by Insights, Planning, and the AI Foundation modules to identify the movement of inventory into a location and to track first/last receipt date attributes.
Adjustments	ADJUSTMENT.csv	Inventory adjustments made at a location, including shrink, wastage, theft, stock counts, and other reasons. Used by Insights and Planning modules.

Table 3-2 (Cont.) Common Foundation Facts

Dimension	Filename(s)	Usage
Purchase Orders	ORDER_HEAD.csv ORDER_DETAIL.csv	The purchase order data for all orders placed with suppliers. Held at a level of order number, supplier, item, location, and date. Separate files are needed for the order attributes and order quantities/amounts. Used by Insights and Planning modules.
Markdowns	MARKDOWN.csv	The currency amount above or below the listed price of an item when that item's price is modified for any reason (planned markdown, POS discount, promotional sale, and so on). Used By Insights and Planning modules.
Transfers	TRANSFER.csv	The movement of inventory between two locations (both physical and virtual). Used By Insights and Planning modules.
Returns to Vendor	RTV.csv	The return of owned inventory to a supplier or vendor. Used by Insights and Planning modules.
Prices	PRICE.csv	The current selling retail value of an item at a location. Used by Insights and AI Foundation modules.
Costs	COST.csv	The base unit cost and derived net costs of an item at a location. Used by Retail Insights only.
Wholesale/Franchise	SALES_WF.csv	Sales and markdown data from wholesale and franchise operations. Used by all modules.
Deal Income	DEAL_INCOME.csv	Income associated with deals made with suppliers and vendors. Used by Insights and Planning modules.

Details on which application modules make use of specific files (or columns within a file) can be found in the Interfaces Guide on My Oracle Support. Make sure you have a full understanding of the data needs for each application you are implementing before moving on to later steps in the process. If it is your first time creating these files, read Data File Generation, for important information about key file structures and business rules that must be followed for each foundation file.

File Upload Samples

When you first upload foundation data into the platform, you will likely provide a small subset of the overall set of files required by your applications. The following examples show a possible series of initial file uploads to help you verify that you are providing the correct sets of data. All dimension files in initialization and history loads must be full snapshots of data; never send partial or incremental files.

Example #1: Calendar Initialization

When you first configure the system you must upload and process the `CALENDAR.csv` file. You will generate the file following the specifications, and also provide a context (ctx) file, as described in Data File Generation.

File to upload: `RAP_DATA_HIST.zip`

Zip file contents:

- `CALENDAR.csv`
- `CALENDAR.csv.ctx`

Example #2: Product and Location Setup

You have finalized the calendar and want to initialize your core product and organization dimensions. You must provide the `PRODUCT.csv` and `ORGANIZATION.csv` files along with their context files, as described in Data File Generation.

File to upload: `RAP_DATA_HIST.zip`

Zip file contents:

- `PRODUCT.csv`
- `PRODUCT.csv.ctx`
- `ORGANIZATION.csv`
- `ORGANIZATION.csv.ctx`

Example #3: Full dimension load

You have a set of dimension files you want to process using the initial dimension load ad hoc processes in POM.

File to upload: `RAP_DATA_HIST.zip`

Zip file contents:

- `PRODUCT.csv`
- `PRODUCT.csv.ctx`
- `ORGANIZATION.csv`
- `ORGANIZATION.csv.ctx`
- `ATTR.csv`
- `ATTR.csv.ctx`
- `PROD_ATTR.csv`
- `PROD_ATTR.csv.ctx`
- `EXCH_RATE.csv`
- `EXCH_RATE.csv.ctx`

Example #4: Sales Data Load

You have finished the dimensions and you are ready to start processing sales history files.

File to upload: `RAP_DATA_HIST.zip`

Zip file contents:

- `SALES.csv`
- `SALES.csv.ctx`

Example #5: Multi-File Fact Data Load

Once you are confident in your data file format and contents, you may send multiple files as separate ZIP uploads for sequential loading in the same run. This process uses a numerical sequence on the end of the ZIP file name. The actual method to loop over these files is to use the `RI_FLOW_ADHOC` standalone flow in the AIF DATA POM schedule, which executes the fact history load for all the standard history load files. Make sure you only schedule a number of recurring flow executions equal to the number of ZIPs you will upload. If you run the flow without any ZIPs remaining to be processed, all jobs will still attempt to execute with 0 rows of data in the staging tables. This will cause the flow to fail as some loads are designed to stop the process if there is no data available.

Files to upload: `RAP_DATA_HIST.zip.1`, `RAP_DATA_HIST.zip.2`, `RAP_DATA_HIST.zip.3`

Zip file contents (in each uploaded zip):

- `SALES.csv`
- `SALES.csv.ctx` – The CTX is only required in the first ZIP file, but it's best to always include it so you can refer to it later in archived files, if needed.

In this example you are loading sales month by month iteratively, but the standalone flow supports all other fact loads as well. You can also combine multiple fact files (for different facts with the same period of time) in each ZIP file upload. Track the status of the files in the `C_HIST_FILES_LOAD_STATUS` table after each cycle execution; it shows whether the file was loaded successfully and how many more files are available to process.

Uploading ZIP Packages

When providing data to the platform, push the compressed files into Object Storage using a ZIP file format. Review the File Transfer Services section for details on how to interact with Object Storage. The ZIP file you use will depend on the data you are attempting to load. The default ZIP file packages are below, but the history ZIP file name is configurable in `C_ODI_PARAM` using parameter name `HIST_ZIP_FILE` if a different one is desired.

Table 3-3 Platform ZIP File Usage

Filenames	Frequency	Notes
RAP_DATA_HIST.zip	Ad Hoc	Used for: • Historical files, such as sales and inventory history for the last 1-2 years. • Loading initial dimensions, such as calendar, merchandise, and location hierarchies prior to history loads. • Initial seeding loads.
RAP_DATA_HIST.zip.1 RAP_DATA_HIST.zip.2 ... RAP_DATA_HIST.zip.N	Ad Hoc / Intraday	Multiple zip uploads are supported for sending historical fact data which should be loaded sequentially. Append a sequence number on the ZIP files starting from 1 and increasing to N, where N is the number of files you are loading. Track the status of the files in `C_HIST_FILES_LOAD_STATUS` table.
RAP_DATA.zip RI_RMS_DATA.zip RI_CE_DATA.zip RI_MFP_DATA.zip RI_EXT_DATA.zip	Daily	Can be used for daily ongoing loads into the platform (for RI and foundation common inputs), and for any daily data going to downstream applications through AIF DATA nightly batch. Different files can be used for different source systems.
RI_REPROCESS_DATA.zip	Ad Hoc	Used to upload individual files which will be appended into an existing nightly batch file set.
ORASE_WEEKLY_ADHOC.zip	Ad Hoc	Used for loading AI Foundation files with ad hoc processes.
ORASE_WEEKLY.zip	Weekly	Used for weekly batch files sent directly to AI Foundation.
ORASE_INTRADAY.zip	Intraday	Used for intraday batch files sent directly to AI Foundation.

Other supported file packages, such as output files and optional input files, are detailed in each module's implementation guides. Except for Planning-specific integrations and customizations (which support additional integration paths and formats), it is expected that all files will be communicated to the platform using one of the filenames above.

Preparing to Load Data

Implementations can follow this general outline for the data load process:

1. Initialize the business and system calendars and configure partitioning parameters, which prepares the database for loading fact data.
2. Load initial dimension data into the dimension and hierarchy tables and perform validation checks on the data from DV/APEX or using RI reports.
3. If implementing any AI Foundation or Planning module, load the dimension data to those systems now. Data might work fine on the input tables but have issues only visible after processing in those systems. Don't start loading history data if your dimensions are not working with all target applications.
4. Load the first set of history files (for example, one month of sales or inventory) and validate the results using DV/APEX.

5. If implementing any AI Foundation or Planning module, stop here and load the history data to those systems as well. Validate that the history data in those systems is complete and accurate per your business requirements.
6. Continue loading history data into RAP until you are finished with all data. You can stop at any time to move some of the data into downstream modules for validation purposes.
7. After history loads are complete, all positional tables, such as Inventory Position, need to be seeded with a full snapshot of source data before they can be loaded using regular nightly batches. This seeding process is used to create a starting position in the database which can be incremented by daily delta extracts. These full-snapshot files can be included in the first nightly batch you run, if you want to avoid manually loading each seed file through one-off executions.
8. When all history and seeding loads are completed and downstream systems are also populated with that data, nightly batches can be started.

Before you begin this process, it is best to prepare your working environment by identifying the tools and connections needed for all your Oracle cloud services that will allow you to interact with the platform, as detailed in Implementation Tools and Data File Generation.

Prerequisites for loading files and running POM processes include:

Prerequisite	Tool / Process
Upload ZIPs to Object Storage	File Transfer Service (FTS) scripts
Invoke adhoc jobs to unpack and load the data	Postman (or similar REST API tool)
Monitor job progress after invoking POM commands	POM UI (Batch Monitoring tab)
Monitoring data loads	APEX / DV (direct SQL queries)

Users must also have the necessary permissions in Oracle Cloud Infrastructure Identity and Access Management (OCI IAM) to perform all the implementation tasks. Before you begin, ensure that your user has at least the following groups (and their `_PREPROD` equivalents if using a stage/dev environment):

Access Needed	Groups Needed
Batch Job Execution	BATCH_ADMINISTRATOR_JOB PROCESS_SERVICE_ADMIN_JOB
Database Monitoring	<tenant ID>-DVContentAuthor (DV) DATA_SCIENCE_ADMINISTRATOR_JOB (APEX)
Retail Home	RETAIL_HOME_ADMIN PLATFORM_SERVICES_ADMINISTRATOR PLATFORM_SERVICES_ADMINISTRATOR_ABSTRACT
RI and AI Foundation Configurations	ADMINISTRATOR_JOB
MFP Configurations	MFP_ADMIN_STAGE / PROD
IPO Configurations	IPO_ADMIN_STAGE / PROD
AP Configurations	AP_ADMIN_STAGE / PROD

Managing System Dates

All Retail Analytics and Planning solutions operate on calendar-based data, but the way the current date is tracked and used is specific to each application due to their unique data needs and business processes. Anytime this document uses terms like "current date" or "today's date", it is not referring to the actual calendar date. These terms refer to the date of the business calendar the application is treating as the current day of processing for any operations it will perform on calendar-based data. Understanding how each application sets and uses this date is critical to working with any RAP solution.

Data Warehouse Current Business Date

Locating the Current Date

The shared data warehouse used for integrating data to all RAP solutions has a specific table used to track the current date, named `W_RTL_CURR_MCAL_G`. This table tracks the current day of the fiscal calendar that all data being processed is relative to, as well as related primary key values used during data loads of calendar information. The most important value in this table is in column `MCAL_NUM` on the row where `MCAL_TYPE = 'DT'`. This date (in `YYYYMMDD` format) is the current business date used by all processes in the AIF DATA batch schedule in POM. Anytime a date value is needed by any job in that schedule, this is the value that will be referenced. This value impacts many of your implementation activities, as detailed below:

- Dimension data, such as the product and location hierarchy, includes fields for the effective start and end dates of each record, and some tables may use the current business date in `W_RTL_CURR_MCAL_G` as the effective start date of the record. That record would then be considered as active and valid only from that date forwards. Attempting to use these records prior to the effective start date will not be allowed by many data warehouse programs.
- Some programs in the AIF DATA batch jobs will behave differently when you are running them for the first time on a given business date, compared to if you re-run them again within the same date. Most programs in the nightly batch are designed to run only once per business date and it is not recommended to run them again for the same date unless Oracle advises it as part of resolving a Service Request.
- Data warehouse table partitions are created by calendar date and are generated for a range of time before and after the current business date in `W_RTL_CURR_MCAL_G`.
- There is a classification of fact data known as positional data, which includes Inventory Positions and Purchase Orders. These datasets are always loaded for the current business date in the nightly batch cycle. All positional datasets load daily data by inserting database records for the current business date from `W_RTL_CURR_MCAL_G`. Positional data cannot be loaded for any other date except when using specific history load programs in ad hoc processes.
- Integrations from the data warehouse to other systems such as Merchandise Financial Planning use the current business date as a point of reference where needed. Positional data like inventory positions is always exported relative to the current business date. Calculated values like the end-of-week date on purchase order amounts will be derived from the current date.

Maintaining the Current Date

The current business date in the data warehouse cannot be changed directly; it requires the use of two specific batch programs named `ETL_BUSINESS_DATE_JOB` and `W_RTL_CURR_MCAL_G_JOB`. The `ETL_BUSINESS_DATE_JOB` will update the value of `MCAL_NUM` using the date you specify as the parameter on the job. `W_RTL_CURR_MCAL_G_JOB` recalculates all of the other fields in the table relative to the new value you set with `ETL_BUSINESS_DATE_JOB`. You will see these two jobs included in many ad hoc process flows in the AIF DATA schedule because it is necessary for you to set or update the current business date as part of that flow.

In the nightly batch, `W_RTL_CURR_MCAL_G_JOB` runs by itself as it will advance the business date by one day when not used alongside `ETL_BUSINESS_DATE_JOB`.

During an implementation, you must update the current business date at the following times:

- Initial dimension load requires that you move the date back into the past, to a point where your historical data will start. This past date must fall within the range of dates provided in your initial calendar file. Having the date in the past ensures the effective start date on dimension records is sufficiently far back to allow for loading fact data on it. You must keep the current business date in the past, preferably not changing it at all, until you finish all history loads.
- When loading seed data for positional facts prior to starting nightly batch, you must set the current business date to match the date in the seed data being loaded.
- Before starting nightly batches, the current business date must be one day prior to the first batch date. This allows the first execution of `W_RTL_CURR_MCAL_G_JOB` in the nightly batch to advance the date to the current date of incoming data.

From an end-user perspective, the current business date in the data warehouse represents the most recent date of data that has been processed by the system. For example, if the incoming data files for the nightly batch contain data for 2024-09-17, then the current business date at the end of that nightly batch run will also be 2024-09-17. This is typically going to be one day behind the actual current date on a calendar, because the data warehouse only processes complete days of information, so it will be current through close-of-business for yesterday's date. This is distinctly different from operational systems like Merchandising Foundation Cloud Services (MFCS), which always wants to be on the actual current date because those systems are working with real-time information. For the scenario where the data warehouse has 2024-09-17 as the current date, MFCS will have 2024-09-18 as the current date at the end of its own batch. Similarly, the date displayed in the POM application is not linked with the date in the data warehouse. After the nightly schedule finishes, the POM date will show today's date. Again, this is one day later than the data warehouse current business date.

Business Date Validation

The AIF DATA nightly batch includes validation of the current business date at two different points. These validations are used to ensure the correct dataset for the current business date is the one being processed, and you have not accidentally started a batch for the wrong date.

The first validation happens in the job `BATCH_VALIDATION_JOB`. It will compare the date currently in the `W_RTL_CURR_MCAL_G` table with the incoming business date on your new data. The date used for comparison will come either as a row on the file `RA_SRC_CURR_PARAM_G.dat` (if it exists) which you would be uploading as part of the nightly batch files, or it will be read directly from the table `RA_SRC_CURR_PARAM_G` when it is being integrated directly from Merchandising Foundation Cloud Service (MFCS). The validation specifically checks that the incoming data is one day later than the current business date and it will fail in all other cases. Using the previous examples, if your current business date is 2024-09-17 in `W_RTL_CURR_MCAL_G` then the incoming date on `RA_SRC_CURR_PARAM_G` must be 2024-09-18, as this represents the date you are now loading data for.

The second validation happens in the job `FACT_POSFACT_VALIDATOR_JOB`. It will compare the date currently in the `W_RTL_CURR_MCAL_G` table (after it has been advanced to the next day) with the dates in the `DAY_DT` column of every positional fact table, such as `W_RTL_INV_IT_LC_DY_FS` and `W_RTL_PRICE_IT_LC_DY_FS`. Positional fact data must always be provided for the current date only because these facts do not support posting to past or future dates. The job will fail if any dates do not exactly match. This validation ensures that you did not accidentally include positional fact data files from a prior date, which can happen if you had some issue producing new data in your source systems and allowed an old file to be uploaded for the next batch run.

AI Foundation Date Usage

AI Foundation applications typically operate on wide ranges of dates rather than one specific date at a time, so they do not always need to know what today's date is. Instead, integrations between the data warehouse and AIF applications will usually be based on the range of dates currently in the data warehouse tables at the time a given job executes, combined with input date parameters on the AIF job itself. For example, the first time you run the sales transaction load using AIF APPS programs, it will identify the range of dates in the data warehouse table and import all of that data in chronological order. When running the program to extract sales in the nightly batch for AIF APPS, it will only extract data that has been created or changed since the last time it ran, so it does not really matter what the current business date in the data warehouse is. If an AIF application does need to manage some form of current business date, it will do so by updating a parameter on the `RSE_CONFIG` table. Only the Inventory Optimization module does this currently, using the parameter `IO_CURRENT_DATE`. If you are ever unsure the date Inventory Optimization is using while executing, you can query this parameter to get that information.

Programs internal to the AIF APPS batch schedule also track the last date they processed data for using a table named `RSE_PROC_STAT`. This table ensures that programs are aware of which dates have already been processed, such that the next time they execute they will look for the next available period of time and not re-run past periods. A common reason to look at the `RSE_PROC_STAT` table is when you encounter an AIF APPS program that appears to do nothing, and you expected it to produce some output. It may not have done any work because it has already run for a certain date, recorded that date in `RSE_PROC_STAT`, and there is no new data available after that date, so the program ends immediately. During an implementation, you may encounter times when you need to delete some dates in `RSE_PROC_STAT` to force a re-run of all past dates of data.

Planning Application Date Usage

Planning applications built on the RPAS platform, such as Merchandise Financial Planning (MFP), have their own concept of current business date centered around a variable named `RPAS_TODAY`. The value of `RPAS_TODAY` is initially managed by the implementation team, as it affects many aspects of the planning application functionality, such as which weeks are considered elapsed or unelapsed in workbooks. The value of `RPAS_TODAY` is updated using the **List/Set/Unset PDS Environment Variables** task in the RPAS online administration tools (OAT). The value of `RPAS_TODAY` is not linked with the data warehouse, nor with AI Foundation applications, so it is remember which dates are being used when integrating data between applications. During implementation, it is expected that `RPAS_TODAY` will be controlled separately from the other systems and they will usually not align until you start running nightly batches.

When using POM for executing batches for a planning application, the value of `RPAS_TODAY` should by unset within OAT, and then it will automatically use the current business date as defined in the POM UI, plus or minus an offset value if you need the application to run for a different date than the date displayed on the POM UI. This process is explained in more detail in the *Retail Predictive Application Server Cloud Edition Administration Guide* chapter "RPASCE Batch Schedule with POM". Only planning applications use the POM date; Retail Insights and AI Foundation applications do not reference the POM date at any time.

Calendar Setup

This is the first step that must be performed in all new environments, including projects that will not be implementing RI, but only AI Foundation or Planning solutions. Before beginning this step, ensure your configurations are complete per the initial configuration sections in the prior chapter. Your `START_DT` and `END_DT` variables must be set correctly for your calendar range (`START_DT` at least 12 months before start of history data) and the `C_MODULE_ARTIFACT` table must have all of the required tables enabled for partitioning. `C_MODULE_EXACT_TABLE` must be configured if you need PLAN partitions for planning data loads.

> **Note:**
>
> In version 25, manual partitioning steps have been deprecated and replaced by automated partitioning. Partition configuration (such as in the `C_MODULE_ARTIFACT` table) are still a required step in the environment setup process, but all other partitioning activities are performed automatically by the fact load programs themselves. You will still see the jobs `CREATE_PARTITION_PRESETUP_JOB`, `ETL_BUSINESS_DATE_JOB`, `W_RTL_CURR_MCAL_G_JOB`, and `CREATE_PARTITION_EXECUTION_JOB` in the process for `CALENDAR_LOAD_ADHOC`, but if they are currently enabled, you may disable them or skip them for new implementations.

1. Upload the calendar file `CALENDAR.csv` (and associated context file) through Object Storage (packaged using the `RAP_DATA_HIST.zip` file).
2. Execute the `HIST_ZIP_FILE_LOAD_ADHOC` process. Example Postman message body:

   ```
   {
   "cycleName":"Adhoc",
   "flowName":"Adhoc",
   "processName":"HIST_ZIP_FILE_LOAD_ADHOC"
   }
   ```

3. Verify that the jobs in the ZIP file load process completed successfully using the POM **Monitoring** screen. Download logs for the tasks as needed for review.
4. Execute the `CALENDAR_LOAD_ADHOC` process. This transforms the calendar data and moves it into all internal data warehouse tables. Ensure that all jobs in this process are enabled in POM before executing it.

   Sample Postman message body:

   ```
   {
     "cycleName":"Adhoc",
     "flowName":"Adhoc",
     "processName":"CALENDAR_LOAD_ADHOC"
   }
   ```

   If you did not disable the manual partitioning jobs as noted above, then you need to use the earlier method of calling this process (which includes partition parameters) or you must add the date values to the job parameters from the POM UI, based on the sample message body below:

```
{
  "cycleName":"Adhoc",
  "flowName":"Adhoc",
  "processName":"CALENDAR_LOAD_ADHOC",

"requestParameters":"jobParams.CREATE_PARTITION_PRESETUP_JOB=2022-02-06,job
Params.ETL_BUSINESS_DATE_JOB=2025-02-01"
}
```

5. If this is your first time loading a calendar file, check the `RI_DIM_VALIDATION_V` view to confirm no warnings or errors are detected. Refer to the *AI Foundation Operations Guide* for more details on the validations performed. The validation job will fail if it doesn't detect data moved to the final table (`W_MCAL_PERIOD_D`). Refer to Sample Validation SQLs for sample queries you can use to check the data.
6. If you need to reload the same file multiple times due to errors, you must Restart the Schedule in POM and then run the ad hoc process `C_LOAD_DATES_CLEANUP_ADHOC` before repeating these steps. This will remove any load statuses from the prior run and give you a clean start on the next execution.

> **Note:**
>
> If any job having `STG` in the name fails during the run, then review the POM logs and it should provide the name of an external LOG or BAD table with more information. These error tables can be accessed from APEX using a support utility. Refer to the *AI Foundation Operations Guide* section on “External Table Load Logs” for the utility syntax and examples.

Once the full process completes successfully, you should verify your data in the core calendar tables in a data warehouse such as `W_MCAL_DAY_D`, `W_MCAL_WEEK_D`, and `W_MCAL_PERIOD_D`. This process is only for populating the data warehouse, it does not move the calendar to the AIF or RPAS applications as there are separate programs for those integrations.

Loading Data from Files

When history and initial seed data comes from flat files, use the following tasks to upload them into RAP:

Table 3-4 Flat File Load Overview

Activity	Description
Initialize Dimensions	Initialize dimensional data (products, locations, and so on) to provide a starting point for historical records to join with. Separate initial load processes are provided for this task.
Load History Data	Run history loads in one or multiple cycles depending on the data volume, starting from the earliest date in history and loading forward to today.

Table 3-4 (Cont.) Flat File Load Overview

Activity	Description
Reloading Dimensions	Reload dimensional data as needed throughout the process to maintain correct key values for all fact data. Dimensional files can be provided in the same package with history files and ad hoc processes run in sequence when loading.
Seed Positional Facts	Seed initial values for positional facts using full snapshots of all active item/locations in the source system. This must be loaded for the date prior to the start of nightly batches to avoid gaps in ongoing data.
Run Nightly Batches	Nightly batches must be started from the business date after the initial seeding was performed.

Completing these steps will load all of your data into the Retail Insights data model, which is required for all implementations. From there, proceed with moving data downstream to other applications as needed, such as AI Foundation modules and Merchandise Financial Planning.

> **Note:**
>
> All steps are provided sequentially, but can be executed in parallel. For example, you may load dimensions into RI, then on to AI Foundation and Planning applications before loading any historical fact data. While historical fact data is loaded, other activities can occur in Planning such as the domain build and configuration updates.

Initialize Dimensions

Loading Dimensions into RI

You cannot load any fact data into the platform until the related dimensions have been processed and verified. The processes in this section are provided to initialize the core dimensions needed to begin fact data loads and verify file formats and data completeness. Some dimensions which are not used in history loads are not part of the initialization process, as they are expected to come in the nightly batches at a later time.

For the complete list of dimension files and their file specifications, refer to the AI Foundation Interfaces Guide on My Oracle Support. The steps below assume you have enabled or disabled the appropriate dimension loaders in POM per your requirements. The process flow examples also assume CSV file usage, different programs are available for legacy DAT files. The *AI Foundation Operations Guide* provides a list of all the job and process flows used by foundation data files, so you can identify the jobs required for your files and disable unused programs in POM.

When you are using RDE jobs to source dimension data from RMFCS and you are not providing any flat files like `PRODUCT.csv`, it is necessary to disable all file-based loaders in the `RI_DIM_INITIAL_ADHOC` process flow from POM. Any job name starting with the following text can be disabled, because RDE jobs will bypass these steps and insert directly to staging tables:

- COPY_SI_
- STG_SI_

- SI_
- STAGING_SI_

1. Disable any dimension jobs you are not using from Batch Administration, referring to the process flows for DAT and CSV files in the *AIF Operations Guide* as needed. If you are not sure if you need to disable a job, it's best to leave it enabled initially. Restart the POM schedule in Batch Monitoring to apply the changes.
2. Provide your dimension files and context files through File Transfer Services (packaged using the `RAP_DATA_HIST.zip` file). All files should be included in a single zip file upload. If you are using data from Merchandising, this is where you should run the RDE `ADHOC` processes such as `RDE_DIM_FLOW_ADHOC`.
3. Execute the `HIST_ZIP_FILE_LOAD_ADHOC` process if you need to unpack a new ZIP file.
4. If you are using DAT file formats (such as when RDE v19 or earlier is used to produce an `RIHIST_RMS_DATA.zip` package) then you need to use the `DAT_REPROCESS_ADHOC` process to import your files and stage them for further processing. For all other use cases, skip this step.
5. Execute the `RI_DIM_INITIAL_ADHOC` process to stage, transform, and load your dimension data from the files. The ETL date on the command should be at a minimum one day before the start of your history load timeframe, but 3-6 months before is ideal. It is best to give yourself a few months of space for reprocessing dimension loads on different dates prior to start of history. Date format is `YYYY-MM-DD`; any other format will not be processed. After running the process, you can verify the dates are correct in the `W_RTL_CURR_MCAL_G` table. If the business date was not set correctly, your data may not load properly.

   Sample Postman message body:

   ```
   {
     "cycleName":"Adhoc",
     "flowName":"Adhoc",
     "processName":"RI_DIM_INITIAL_ADHOC",
     "requestParameters":"jobParams.ETL_BUSINESS_DATE_JOB=2017-12-31"
   }
   ```

   > **Note:**
   >
   > If any job having `STG` in the name fails during the run, then review the POM logs and it should provide the name of an external LOG or BAD table with more information. These error tables can be accessed from APEX using a support utility. Refer to the *AI Foundation Operations Guide* section on "External Table Load Logs" for the utility syntax and examples.

If this is your first dimension load, you will want to validate the core dimensions such as product and location hierarchies using APEX. Refer to Sample Validation SQLs for sample queries you can use for this.

If any jobs fail during this load process, you may need to alter one or more dimension data files, re-send them in a new zip file upload, and re-execute the programs. Only after all core dimension files have been loaded (`CALENDAR`, `PRODUCT`, `ORGANIZATION`, and `EXCH_RATE`) can you proceed to history loads for fact data. Make sure to query the `RI_DIM_VALIDATION_V` view for any warnings/errors after the run. Refer to the *AI Foundation Operations Guide* for more details on the validation messages that may occur. This view primarily uses the table

C_DIM_VALIDATE_RESULT, which can be separately queried instead of the view to see all the columns available on it.

If you need to reload the same file multiple times due to errors, you must Restart the Schedule in POM and then run the ad hoc process C_LOAD_DATES_CLEANUP_ADHOC before repeating these steps. This will remove any load statuses from the prior run and give you a clean start on the next execution.

> **Note:**
>
> Starting with version 23.1.101.0, the product and organization file loaders have been redesigned specifically for the initial ad hoc loads. In prior versions, you must not reload multiple product or organization files for the same ETL business date, as it treats any changes as a reclassification and can cause data issues while loading history. In version 23.x, the dimensions are handled as “Type 1” slowly changing dimensions, meaning the programs do not look for reclasses and instead perform simple merge logic to apply the latest hierarchy data to the existing records, even if levels have changed.

As a best practice, you should disable all POM jobs in the RI_DIM_INITIAL_ADHOC process except the ones you are providing new files for. For example, if you are loading the PRODUCT, ORGANIZATION, and EXCH_RATE files as your dimension data for AI Foundation, then you could just execute the set of jobs for those files and disable the others. Refer to the *AI Foundation Operations Guide* for a list of the POM jobs involved in loading each foundation file, if you wish to disable jobs you do not plan to use to streamline the load process.

Hierarchy Deactivation

Foundation dimension ad hoc loads are treated differently than the nightly batch programs specifically for product and location data. They use Type 1 slowly-changing dimension (SCD) behavior, which means that the system will not create new records every time a parent/child relationship changes (also known as reclassification). Instead, they will perform a simple merge on top of existing data to maintain as-is hierarchy definitions. If you change hierarchy classifications on items or locations from one dimension load to the next, the old classifications are overwritten and lost.

The foundation data model holds hierarchy records separately from product data, so it is also necessary to perform maintenance on hierarchies to maintain a single active set of records that should be propagated downstream to other RAP applications. This maintenance is performed using the program W_PROD_CAT_DH_CLOSE_JOB in the RI_DIM_INITIAL_ADHOC process. The program will detect unused hierarchy nodes which have no children after the latest data has been loaded into W_PROD_CAT_DH and it will close them (set to CURRENT_FLG=N). This is required because, in the data model, each hierarchy level is stored as a separate record, even if that level is not being used by any products on other tables. Without the cleanup activity, unused hierarchy levels would accumulate in W_PROD_CAT_DH and be available in AI Foundation, which is generally not desired.

There are some scenarios where you may want to disable this program. For example, if you know the hierarchy is going to change significantly over a period of time and you don’t want levels to be closed and re-created every time a new file is loaded, you must disable W_PROD_CAT_DH_CLOSE_JOB. You can re-enable it later and it will close any unused levels that remain after all your changes are processed. Also be aware that the program is part of the nightly batch process too, so once you switch from historical to nightly loads, this job will be enabled and will close unused hierarchy levels unless you intentionally disable it. This job must be disabled if you are using RDE programs to load Merchandising data from Merchandising Foundation Cloud Services (MFCS), because the hierarchy is automatically maintained without using this job..

Loading Dimensions to Other Applications

Once you have successfully loaded dimension data, you should pause the dataload process and push the dimensions to AI Foundation Cloud Services and the Planning Data Store (if applicable). This allows for parallel data validation and domain build activities to occur while you continue loading data. Review sections Sending Data to AI Foundation and Sending Data to Planning for details on the POM jobs you may execute for this.

The main benefits of this order of execution are:

1. Validating the hierarchy structure from the AI Foundation interface provides an early view for the customer to see some application screens with their data.
2. Planning apps can perform the domain build activity without waiting for history file loads to complete, and can start to do other planning implementation activities in parallel to the history loads.
3. Data can be made available for custom development or validations in Innovation Workbench.

Do not start history loads for facts until you are confident all dimensions are working throughout your solutions. Once you begin loading facts, it becomes much harder to reload dimension data without impacts to other areas. For example, historical fact data already loaded will not be automatically re-associated with hierarchy changes loaded later in the process.

Load History Data

Historical fact data is a core foundational element to all solutions in Retail Analytics and Planning. As such, this phase of the implementation can take the longest amount of time during the project, depending on the volumes of data, the source of the data, and the amount of transformation and validation that needs to be completed before and after loading it into the Oracle database.

It is important to know where in the RAP database you can look to find what data has been processed, what data may have been rejected or dropped due to issues, and how far along in the overall load process you are. The following tables provide critical pieces of information throughout the history load process and can be queried from APEX.

Table 3-5 Inbound Load Status Tables

Table	Usage
C_HIST_LOAD_STATUS	Tracks the progress of historical ad hoc load programs for inventory and pricing facts. This table will tell you which Retail Insights tables are being populated with historical data, the most recent status of the job executions, and the most recently completed period of historical data for each table. Use APEX or Data Visualizer to query this table after historical data load runs to ensure the programs are completing successfully and processing the expected historical time periods.
C_HIST_FILES_LOAD_STATUS	Tracks the progress of zip file processing when loading multiple files in sequence using scheduled standalone process flows.

Table 3-5 (Cont.) Inbound Load Status Tables

Table	Usage
C_LOAD_DATES	Tracks the status of jobs run from the AIF DATA schedule in POM that involve extracting, transforming, or loading data. When a job has an entry in this table, restarting it will behave differently from the initial run. Restarting a job with a “Success” status will skip the job and do nothing. Restarting a job with “Failed” status will attempt to re-run the job.
C_LOAD_DATES_DETAILS	Tracks the individual steps of a program in `C_LOAD_DATES` when task-level restartability is enabled. Managing the steps of a program on this table allows for a failed run of a job to restart from the step where it failed, instead of starting from the beginning of the program. If you need to start a program over from the beginning, you would have to purge all the records from both `C_LOAD_DATES` and `C_LOAD_DATES_DETAILS` tables.
W_ETL_REJECTED_RECORDS	Summary table capturing rejected fact record counts that do not get processed into their target tables in Retail Insights. Use this to identify other tables with specific rejected data to analyze. Does not apply to dimensions, which do not have rejected record support at this time.
E$_W_RTL_SLS_TRX_IT_LC_DY_TMP	Example of a rejected record detail table for Sales Transactions. All rejected record tables start with the `E$_` prefix. These tables are created at the moment the first rejection occurs for a load program. `W_ETL_REJECTED_RECORDS` will tell you which tables contain rejected data for a load. These tables may not initially be granted to APEX for you to read from. To grant access, run the `RABE_GRANT_ACCESS_TO_IW_ADHOC_PROCESS` ad hoc process in the AIF APPS schedule in POM. This will allow you to select from these error tables to review rejection details.

When loading data from flat files for the first time, it is common to have bad records that cannot be processed by the RAP load procedures, such as when the identifiers on the record are not present in the associated dimension tables. The foundation data loads leverage rejected record tables to capture all such data so you can see what was dropped by specific data load and needs to be corrected and reloaded. These tables do not exist until rejected records occur during program execution, and are not initially granted to APEX unless you have run `RABE_GRANT_ACCESS_TO_IW_ADHOC_PROCESS`. Periodically monitor these tables for rejected data which may require reloading.

The overall sequence of files to load will depend on your specific data sources and conversion activities, but the recommendation is listed below as a guideline.

1. **Sales** – Sales transaction data is usually first to be loaded, as the data is critical to running most applications and needs the least amount of conversion.
2. **Inventory Receipts** – If you need receipt dates for downstream usage, such as in Lifecycle Pricing Optimization, then you need to load receipt transactions in parallel with Inventory Positions. For each file of receipts loaded, also load the associated inventory positions afterwards.
3. **Inventory Position** – The main stock-on-hand positions file is loaded next. This history load also calculates and stores data using the receipts file, so `INVENTORY.csv` and `RECEIPT.csv` must be loaded at the same time, for the same periods.

4. **Pricing** – The price history file is loaded after sales and inventory are complete because many applications need only the first two datasets for processing. Potentially, price history may also be the largest volume of data; so it's good to be working within your other applications in parallel with loading price data.
5. **All other facts** – There is no specific order to load any of the other facts like transfers, adjustments, markdowns, costs, and so on. They can be loaded based on your downstream application needs and the availability of the data files.

For your first time implementing this history load process, you may also leverage the reference paper and scripts in My Oracle Support (Doc ID 2539848.1) titled AI Foundation Historical Data Load Monitoring. This document will guide you through one way you can monitor the progress of history loads, gather statistics on commonly used tables, and verify that data is moving from the input tables to the target tables in the database.

Automated History Loads

Once you have performed your history loads manually a couple of times (following all steps in later sections) and validated the data is correct, you may wish to automate the remaining file loads. A standalone process flow `RI_FLOW_ADHOC` is available in POM that can run the fact history loads multiple times using your specified start times. Follow the steps below to enable this process:

1. Upload multiple ZIP files using FTS, each containing one set of files for the same historical period. Name the files like `RAP_DATA_HIST.zip`, `RAP_DATA_HIST.zip.1`, `RAP_DATA_HIST.zip.2` and so on, incrementing the index on the end of the zip file name after the first one.
2. In the POM batch administration screen, ensure all of the jobs in the `RI_FLOW_ADHOC` are enabled, matching your initial fact history standalone runs. The following processes in the process flow are common to all runs and all jobs should be enabled in them:
   - FLOW_LOAD_START_PROCESS
   - FLOW_LOAD_END_PROCESS
   - CLEANUP_C_LOAD_DATES_FLOW_PROCESS
   - ZIP_FILE_LOAD_FLOW_PROCESS
   - FACT_FLOW_END_PROCESS
3. Schedule the standalone flows from Scheduler Administration to occur at various intervals throughout the day. Space out the cycles based on how long it took to process your first file.
4. Monitor the load progress from the Batch Monitoring screen to see the results from each run. Monitor the table `C_HIST_FILES_LOAD_STATUS` to verify the ZIP files you uploaded were all processed successfully. Validate that data is being loaded successfully in your database periodically throughout the standalone runs. If a run fails for any reason, it will not allow more runs to proceed until the issue is resolved.

Sales History Load

RAP supports the loading of sales transaction history using actual transaction data or daily/weekly sales totals. If loading data at an aggregate level, all key columns (such as the transaction ID) are still required to have some value. The sales data may be provided for a range of dates in a single file. The data should be loaded sequentially from the earliest week to the latest week but, unlike inventory position, you may have gaps or out-of-order loads, because the data is not stored positionally. Refer to Data File Generation for more details on the file requirements.

> **Note:**
>
> Many parts of AI Foundation require transactional data for sales, so loading aggregate data should not be done unless you have no better alternative.

If you are not loading sales history for Retail Insights specifically, then there are many aggregation programs that can be disabled in the POM standalone process. Most aggregation programs (jobs ending in `_A_JOB`) populate additional tables used only in BI reporting. The following list of jobs must be enabled in the `HIST_SALES_LOAD_ADHOC` process to support AIF and Planning data needs, but all others can be disabled for non-RI projects:

- VARIABLE_REFRESH_JOB
- ETL_REFRESH_JOB
- W_EMPLOYEE_D_JOB
- SEED_EMPLOYEE_D_JOB
- W_PARTY_PER_D_JOB
- SEED_PARTY_PER_D_JOB
- W_RTL_CO_HEAD_D_JOB
- W_RTL_CO_LINE_D_JOB
- SEED_CO_HEAD_D_JOB
- SEED_CO_LINE_D_JOB
- W_RTL_SLS_TRX_IT_LC_DY_F_JOB
- RA_ERROR_COLLECTION_JOB
- RI_GRANT_ACCESS_JOB
- RI_CREATE_SYNONYM_JOB
- ANAYLZE_TEMP_TABLES_JOB
- W_RTL_SLS_IT_LC_DY_TMP_JOB
- W_RTL_SLS_IT_LC_WK_A_JOB
- W_RTL_PROMO_D_TL_JOB
- SEED_PROMO_D_TL_JOB
- W_PROMO_D_RTL_TMP_JOB
- W_RTL_SLSPR_TRX_IT_LC_DY_F_JOB
- W_RTL_SLSPK_IT_LC_DY_F_JOB
- W_RTL_SLSPK_IT_LC_WK_A_JOB
- REFRESH_RADM_JOB

The other process used, `HIST_STG_CSV_SALES_LOAD_ADHOC`, can be run with all jobs enabled, as it is only responsible for staging the files in the database. Make sure to check the enabled jobs in both processes before continuing.

If you will be loading extremely large volumes of data in a single file (100M+ records), then you may encounter issues where the POM job to load the file completes before the file is finished importing. This is due to a default maximum runtime of 3600 seconds on these types of POM jobs. You may increase the maximum runtime by editing the job parameters of `COPY` and `STG` job types to append the parameter: `maxRunTime=12345`. The numeric value should be the number of seconds you expect the process to run for. For example, locate the jobs `COPY_SI_SALES_JOB` and `STG_SI_SALES_JOB` in the process `HIST_STG_CSV_SALES_LOAD_ADHOC`, then edit the parameters to look like the below:

```
disJobType=OBJ_STORE_PUT||object=ri/data/staging/SALES.csv||
objectFilepath=/u01/retail/ri/mmhome/data/staging/SALES.csv|| maxRunTime=25200
disJobType=EXT_TBL_LOAD||tblName=W_RTL_SLS_TRX_IT_LC_DY_FTXS||fileName=ri/
data/staging/SALES.csv|| maxRunTime=25200
```

Always edit the parameters from Batch Administration and then restart your POM schedule to apply the changes in Batch Monitoring.

After confirming the list of enabled sales jobs and applying any necessary configuration changes, perform the following steps:

1. Create the file `SALES.csv` containing one or more days of sales data along with a CTX file defining the columns which are populated. Optionally include the `SALES_PACK.csv` file as well.
2. Upload the history files to Object Storage using the `RAP_DATA_HIST.zip` file.
3. Execute the `HIST_ZIP_FILE_LOAD_ADHOC` process.
4. Execute the `HIST_STG_CSV_SALES_LOAD_ADHOC` process to stage the data in the database. Validate your data before proceeding. Refer to Sample Validation SQLs for sample queries you can use for this.
5. Execute the `HIST_SALES_LOAD_ADHOC` batch processes to load the data. If no data is available for certain dimensions used by sales, then the load process can seed the dimension from the history file automatically. Enable seeding for all of the dimensions according to the initial configuration guidelines; providing the data in other files is optional.

   Several supplemental dimensions are involved in this load process, which may or may not be provided depending on the data requirements. For example, sales history data has promotion identifiers, which would require data on the promotion dimension.

   Sample Postman message bodies:

   ```
   {
     "cycleName":"Adhoc",
     "flowName":"Adhoc",
     "processName":"HIST_STG_CSV_SALES_LOAD_ADHOC"
   }

   {
     "cycleName":"Adhoc",
     "flowName":"Adhoc",
     "processName":"HIST_SALES_LOAD_ADHOC"
   }
   ```

> **Note:**
>
> If any job having `STG` in the name fails during the run, then review the POM logs and it should provide the name of an external LOG or BAD table with more information. These error tables can be accessed from APEX using a support utility. Refer to the *AI Foundation Operations Guide* section on “External Table Load Logs” for the utility syntax and examples.

After the load is complete, you should check for rejected records, as this will not cause the job to fail but it will mean not all data was loaded successfully. Query the table `W_ETL_REJECTED_RECORDS` from IW to see a summary of rejections. If you cannot immediately identify the root cause (for example, missing products or locations causing the data load to skip the records) there is a utility job `W_RTL_REJECT_DIMENSION_TMP_JOB` that allows you analyze the rejections for common reject reasons. Refer to the *AIF Operations Guide* for details on configuring and running the job for the first time if you have not used it before.

This process can be repeated as many times as needed to load all history files for the sales transaction data. If you are sending data to multiple RAP applications, do not wait until all data files are processed to start using those applications. Instead, load a month or two of data files and process them into all apps to verify the flows before continuing.

> **Note:**
>
> Data cannot be reloaded for the same records multiple times, as sales data is treated as additive. If data needs correction, you must post only the delta records (for example, send -5 to reduce a value by 5 units) or erase the table and restart the load process using `RI_SUPPORT_UTIL` procedures in APEX. Raise a Service Request with Oracle if neither of these options resolve your issue.

Once you have performed the load and validated the data one time, you may wish to automate the remaining file loads. A standalone process flow `RI_FLOW_ADHOC` is available in POM that can run the sales history load multiple times using your specified start times. Follow the steps below to leverage this process:

1. Upload multiple ZIP files each containing one `SALES.csv` and naming them as `RAP_DATA_HIST.zip`, `RAP_DATA_HIST.zip.1`, `RAP_DATA_HIST.zip.2` and so on, incrementing the index on the end of the zip file name. Track the status of the files in the `C_HIST_FILES_LOAD_STATUS` table once they are uploaded and at least one execution of the `HIST_ZIP_FILE_UNLOAD_JOB` process has been run.
2. In the POM batch administration screen, ensure all of the jobs in the `RI_FLOW_ADHOC` are enabled, matching your initial ad hoc run. Schedule the standalone flows from Scheduler Administration to occur at various intervals throughout the day. Space out the runs based on how long it took to process your first file.
3. Monitor the load progress from the Batch Monitoring screen to see the results from each run cycle.

Inventory Position History Load

RAP supports the loading of inventory position history using either full end-of-week snapshots or an initial snapshot followed by end-of-week incremental updates.

When using full weekly snapshots, the data may be provided one week at a time or as multiple weeks in a single file. The data must be loaded sequentially from the earliest week to the latest week with no gaps or out-of-order periods. For example, you cannot start with the most recent inventory file and go backward; you must start from the first week of history. Refer to Data File Generation for more details on the file requirements.

When using the initial snapshot followed by incremental updates, you may be able to provide the entire history as one file. The first week of inventory data in the file must be a complete snapshot of all the inventoried item/locations. For every week-ending date after the first, you must only provide the changed records relative to the prior weeks. This includes changes to the inventory where the balance drops to zero for that week. If you still need to split this data across multiple files, the first week in every file must be a full snapshot to establish starting positions for the incremental updates. Using this approach to loading history, the incremental data is applied to the full snapshot and the final end-of-week positions will be all new records in the current week plus unchanged records from the prior weeks. This greatly reduces the size of your history file, but requires you to extract incremental updates from the source system, which is not always possible. If you want to use the incremental inventory history load method, update the parameter `RI_INV_HIST_LOAD_INC` to a value of `Y` in `C_ODI_PARAM_VW` table in the AI Foundation Control & Tactical Center. The default value is `N`, meaning only full snapshot data is expected.

A variety of `C_ODI_PARAM_VW` settings are also available in the Control Center to disable inventory features that are not required for your implementation. All of the following parameters can be changed to a value of `N` during the history load and enabled later for daily batches, as it will greatly improve the load times:

- `RI_INVAGE_REQ_IND` – Disables calculation of first/last receipt dates and inventory age measures. Receipt date calculation is used in RI and required for Lifecycle Pricing Optimization (as a method of determining entry/exit dates for items). It is also required for Forecasting for the Short Lifecycle (SLC) methods. Set to `Y` if using any of these applications.
- `RA_CLR_LEVEL` – Disables the mapping of clearance event IDs to clearance inventory updates. Used only in RI reporting.
- `RI_PRES_STOCK_IND` – Disables use of replenishment data for presentation stock to calculate inventory availability measures. Used only in RI reporting.
- `RI_BOH_SEEDING_IND` – Disables the creation of initial beginning-on-hand records so analytics has a non-null starting value in the first week. Used only in RI reporting.
- `RI_MOVE_TO_CLR_IND` – Disables calculation of move-to-clearance inventory measures when an item/location goes into or out of clearance status. Used only in RI reporting.
- `RI_MULTI_CURRENCY_IND` – Disables recalculation of primary currency amounts if you are only using a single currency. Should be enabled for multi-currency, or disabled otherwise.

If you will be loading inventory history after you have already started nightly batches, then you must also update two additional parameters:

- `INV_NIGHTLY_BATCH_IND` – Change this to `Y` to indicate that nightly batches have been run but you are planning to load history for prior dates
- `INV_LAST_HIST_LOAD_DT` – Set this to the final week of history data you plan to load, which must be a week-ending date and it must be before the nightly batches were started

Although it is supported, it is not advisable to load history data after nightly batches have started. It would be difficult to erase or correct historical data after it is loaded without affecting your nightly batch data as well. For this reason it is best to validate the history data thoroughly in a non-production environment before loading it to the production system.

If you will be loading extremely large volumes of data in a single file (100M+ records), then you may encounter issues where the POM job to load the file completes before the file is finished importing. This is due to a default maximum runtime of 3600 seconds on these types of POM jobs. You may increase the maximum runtime by editing the job parameters of `COPY` and `STG` job types to append the parameter: `maxRunTime=12345`. The numeric value should be the number of seconds you expect the process to run for. For example, locate the jobs `COPY_SI_INVENTORY_JOB` and `STG_SI_INVENTORY_JOB` in the process `HIST_STG_CSV_INV_LOAD_ADHOC`, then edit the parameters to look like the below:

```
disJobType=OBJ_STORE_PUT||object=ri/data/staging/INVENTORY.csv||
objectFilepath=/u01/retail/ri/mmhome/data/staging/INVENTORY.csv||
maxRunTime=25200
disJobType=EXT_TBL_LOAD||tblName=W_RTL_INV_IT_LC_DY_FTXS||fileName=ri/data/
staging/INVENTORY.csv|| maxRunTime=25200
```

Always edit the parameters from Batch Administration and then restart your POM schedule to apply the changes in Batch Monitoring.

After confirming the list of enabled inventory jobs and applying any necessary configuration changes, perform the following steps:

1. If you need inventory to keep track of First/Last Receipt Dates for use in Lifecycle Pricing Optimization or Forecasting (SLC) then you must first load a `RECEIPT.csv` file for the same historical period as your inventory file (because it is used in forecasting, that may make it required for your Inventory Planning Optimization loads as well, if you plan to use SLC forecasting). You must also set `RI_INVAGE_REQ_IND` to `Y`. Receipts are loaded using the process `HIST_CSV_INVRECEIPTS_LOAD_ADHOC`. Receipts may be provided at day or week level depending on your history needs.
2. Create the file `INVENTORY.csv` containing one or more weeks of inventory snapshots in chronological order along with your CTX file to define the columns that are populated. The `DAY_DT` value on every record must be an end-of-week date (Saturday by default).
3. Optionally, also include the `INVENTORY_PACK.csv` file for pack-level inventory history. The main inventory file must only contain regular items and pack component items; it should not contain pack items. The `INVENTORY_PACK.csv` file is for pack item inventory positions, which are typically held only at warehouses. Pack-level inventory history is only relevant for Retail Insights reporting. Inventory Planning Optimization (IPO) only requires the current inventory positions for pack inventory at the warehouses; it does not need the history.
4. Upload the history file and its context file to Object Storage using the `RAP_DATA_HIST.zip` file.
5. Update column `HIST_LOAD_LAST_DATE` on the table `C_HIST_LOAD_STATUS` to be the date matching the last day of your overall history load (may be later than the dates in the current file). This can be done from the Control & Tactical Center. If you are loading history after your nightly batches were already started, then you must set this date to be the last week-ending date <u>before</u> your first daily/weekly batch. No other date value can be used in this case.
6. Execute the `HIST_ZIP_FILE_LOAD_ADHOC` process.
7. If you are providing `RECEIPT.csv` for tracking receipt dates in history, run `HIST_CSV_INVRECEIPTS_LOAD_ADHOC` at this time.
8. Execute the `HIST_STG_CSV_INV_LOAD_ADHOC` process to stage your data into the database. Validate your data in `W_RTL_INV_IT_LC_DY_FS` (and optionally `W_RTL_INVPK_IT_LC_DY_FS` if

using packs) before proceeding. Refer to Sample Validation SQLs for sample queries you can use for this.

9. Execute the `HIST_INV_LOAD_ADHOC` batch process to load the file data. The process loops over the files one week at a time until all weeks are loaded. It updates the `C_HIST_LOAD_STATUS` table with the progress, which you can monitor from APEX or DV. Sample Postman message bodies:

```
{
  "cycleName":"Adhoc",
  "flowName":"Adhoc",
  "processName":"HIST_STG_CSV_INV_LOAD_ADHOC"
}

{
  "cycleName":"Adhoc",
  "flowName":"Adhoc",
  "processName":"HIST_INV_LOAD_ADHOC"
}
```

This process can be repeated as many times as needed to load all history files for the inventory position. Remember that inventory cannot be loaded out of order, and you cannot go back in time to reload files after you have processed them (for the same item/loc intersections). If you load a set of inventory files and then find issues during validation, erase the tables in the database and restart the load with corrected files.

If you finish the entire history load and need to test downstream systems (like Inventory Planning Optimization) then you must populate the table `W_RTL_INV_IT_LC_G` first (the history load skips this table). There is a separate standalone job `HIST_LOAD_INVENTORY_GENERAL_JOB` in the process `HIST_INV_GENERAL_LOAD_ADHOC` that you may execute to copy the final week of inventory from the fact table to this table. IPO also has the ability to use the last available week from `W_RTL_INV_IT_LC_WK_A` if you are not ready to populate the `W_RTL_INV_IT_LC_G` table with your final positional data.

If your inventory history has invalid data, you may get rejected records and the batch process will fail with a message that rejects exist in the data. If this occurs, you cannot proceed until you resolve your input data, because rejections on positional data MUST be resolved for one date before moving onto the next. If you move onto the next date without reprocessing any rejected data, that data is lost and cannot be loaded at a later time without starting over. When this occurs:

1. The inventory history load will automatically populate the table `W_RTL_REJECT_DIMENSION_TMP` with a list of invalid dimensions it has identified. If you are running any other jobs besides the history load, you can also run the process `W_RTL_REJECT_DIMENSION_TMP_ADHOC` to populate that table manually. You have the choice to fix the data and reload new files or proceed with the current file

2. After reviewing the rejected records, run `REJECT_DATA_CLEANUP_ADHOC`, which will erase the `E$` table and move all rejected dimensions into a skip list. You must pass in the module code you want to clean up data for as a parameter on the POM job (in this case the module code is `INV`). The skip list is loaded to the table `C_DISCARD_DIMM`. Skipped identifiers will be ignored for the current file load, and then reset for the start of the next run.

   Example Postman message body:

```
{
"cycleName": "Adhoc",
```

```
"flowName":"Adhoc",
"processName":"REJECT_DATA_CLEANUP_ADHOC",
"requestParameters":"jobParams.REJECT_DATA_CLEANUP_JOB=INV"
}
```

3. If you want to fix your files instead of continuing the current load, stop here and reload your dimensions and/or fact data following the normal process flows.
4. If you are resuming with the current file with the intent to skip all data in `C_DISCARD_DIMM`, restart the failed POM job now. The skipped records are permanently lost and cannot be reloaded unless you erase your inventory data and start loading files from the beginning.

Log a Service Request with Oracle Support for assistance with any of the above steps if you are having difficulties with loading inventory history or dealing with rejected records.

Reloading Inventory Data

If you need to reload new inventory data from the beginning, you must first erase the inventory fact tables and clean up the configurations. Use the Data Cleanup Utility described in the *AIF Operations Guide* to truncate all inventory fact tables. You should use the subject area cleanup for this purpose, passing in a value of **Inventory Position** as the subject area name. This utility when run for the inventory subject area will truncate the tables, reset the columns in `C_HIST_LOAD_STATUS`, and cleanup metadata relating to the partitions to prepare it for a fresh load of data. It will not truncate `TMP` tables, as those will be truncated automatically when the history load programs are executed.

Make sure you check `C_HIST_LOAD_STATUS` before starting a new inventory load as you may want to enable different tables or change the `HIST_LOAD_LAST_DATE` values to align with your new data. Verify that the `MAX_COMPLETED_DATE` and `HIST_LOAD_STATUS` columns are null for all rows you will be reprocessing.

Price History Load

Certain applications, such as Lifecycle Pricing Optimization, require price history to perform their calculations. Price history is similar to inventory in that it is a positional, but it can be loaded in a more compressed manner due to the extremely high data volumes involved. The required approach for price history is as follows:

1. Update `C_HIST_LOAD_STATUS` for the `PRICE` records in the table, specifying the last date of history load, just as you did for inventory. For whatever value you put in the `HIST_LOAD_LAST_DATE` column, ensure that you load price data up to and including that date. There is logic in the history load to refresh the `W_RTL_PRICE_IT_LC_G` table; that logic only runs on the final day of historical processing. But, if you don't load files all the way up to that date, it will never run and you may have incorrect data as a result. If you are loading history after your nightly batches already started, then you must set this date to be the last week-ending date <u>before</u> your first daily/weekly batch. No other date value can be used in this case.
2. Update `C_ODI_PARAM` parameters based on your implementation plans:
   a. Set parameter `PRICE_LAST_HIST_LOAD_DT` to equal the final date of history processing. It should match the value used on `C_HIST_LOAD_STATUS` for `HIST_LOAD_LAST_DATE`. This parameter sets a threshold after which time only the nightly batch job may be used to load price data. The history load program will not be allowed to load data past this date.

b. If you are loading prices for LPO applications specifically, then update parameter RI_LAST_MKDN_HIST_IND to have a value of Y. This will populate some required fields for LPO markdown and promotion price history.

c. If you are loading price data in multiple phases (with each phase of history load having a different set of products and locations) then you must maintain parameter HIST_LOAD_PHASE to represent which load cycle you are in, using values such as PHASE1 for the first load phase, PHASE2 for the second load phase, and so on.

3. Create an initial, full snapshot of price data in PRICE.csv for the first day of history to establish starting positions for all item/locations. All initial price records must come with a type code of 0. You could load this initial date by itself or as part of a larger file having multiple days of data.

4. Create additional PRICE files containing just price changes for a period of time (such as a month) with the appropriate price change type codes and effective day dates for those changes. Load each file one at a time using the history processes.

5. The history procedure will iterate over the provided files day by day, starting from the first day of history, up to the last historical load date specified in C_HIST_LOAD_STATUS for the pricing fact. For each date, the procedure checks the staging data for effective price change records and loads them, then moves on to the next date.

The process to perform price history loads is similar to the inventory load steps. It uses the PRICE.csv file and the HIST_CSV_PRICE_LOAD_ADHOC process (the price load only has one load process instead of two like sales/inventory). Just like inventory, you must load the data sequentially; you cannot back-post price changes to earlier dates than what you have already loaded. Refer to Data File Generation for complete details on how to build this file.

Just like inventory, the REJECT_DATA_CLEANUP_ADHOC process may be used when records are rejected during the load. Price loads cannot continue until you review and clear the rejections.

```
{
  "cycleName": "Adhoc",
  "flowName":"Adhoc",
  "processName":"REJECT_DATA_CLEANUP_ADHOC",
  "requestParameters":"jobParams.REJECT_DATA_CLEANUP_JOB=PRICE"
}
```

If you will not be loading a complete history file, or you want to skip history and run price seeding or nightly cycles, then you must be aware of the nightly job behavior as it relates to history. The nightly batch job W_RTL_PRICE_IT_LC_DY_F_JOB has a validation rule that looks for history data and fails the batch if it is not found. It is looking specifically in the table W_RTL_PRICE_IT_LC_G, because this table must be populated when the price history job runs for the last date (the value in HIST_LOAD_LAST_DATE). The reason for this validation is that the system calculates many additional fields like LAST_MKDN_DT that need to include your historical price activity. If you run the nightly batch jobs without populating W_RTL_PRICE_IT_LC_G, all calculated fields will start as null or having only the current night's price data. If you are not planning to load complete price history up to the value on HIST_LOAD_LAST_DATE, then you may disable this validation rule from C_ODI_PARAM_VW in the Manage System Configurations screen. Look for the RI_CHK_PRICE_G_EMPTY_IND parameter and update the value to N. This will allow W_RTL_PRICE_IT_LC_DY_F_JOB to complete even if no history has been loaded and the W_RTL_PRICE_IT_LC_G table is empty.

Reloading Price Data

If you need to reload new price data for the same items and locations already loaded, you must first erase the pricing fact tables and clean up the configurations. There are two methods available for this and both should be used as of the current release:

1. Use the Data Cleanup Utility found in the *AIF Operations Guide* to truncate all price tables. You may use the subject area cleanup for this purpose, passing in a value of `Price` as the subject area name. This is required in order to efficiently delete any large volume tables, which performs better from this utility.
2. Use the history cleanup standalone process `HIST_DATA_CLEANUP_ADHOC`, which has only one job named `HIST_DATA_CLEANUP_JOB`. This job requires a module code as an input parameter, so pass in a value of `PRICE` to erase price data. This job will also take care of resetting `C_HIST_LOAD_STATUS` data for pricing loads. This can run after the first step and will take care of price history `TMP` table cleanup that the support utility does not do.

If you will be loading price data for new items or locations on top of existing history data instead of starting over, you need to update the `C_ODI_PARAM` parameter `HIST_LOAD_PHASE` to a different value to indicate a new load is being performed that should not conflict with the existing data. The default value when you do your first history load is `PHASE1`, so update it to `PHASE2` for your second load, `PHASE3` for your third load, and so on. Each history load phase must have a distinct set of item/locations that do not already have data in the price fact. The unique phase identifier will ensure the price fact quarterly seeding logic only applies to the item/locations in that phase.

Make sure you check `C_HIST_LOAD_STATUS` before starting a new price load as you may want to enable different tables or change the `HIST_LOAD_LAST_DATE` values to align with your new data. Verify that the `MAX_COMPLETED_DATE` and `HIST_LOAD_STATUS` columns are null for all rows you will be processing from the start of history.

Purchase Order Loads

Purchase Order data does not support loading of historical records, as all usage of PO data is for forward-looking calculations, such as open on-order amounts. However, it is common to need some test data to be loaded prior to nightly batches so that you can validate the purchase order calculations in your Planning solutions during implementation. When loading PO data, make sure you develop the files following the guidance in the data files generation section of this document and in the RAP interfaces guide.

1. Create the files `ORDER_HEAD.csv` and `ORDER_DETAIL.csv` containing a full snapshot of purchase order data for a single date. The `DAY_DT` value on every record in `ORDER_DETAIL.csv` must be the same and represent the current business date in RAP at the time the file will be loaded. The load will fail for any other value of `DAY_DT` besides the current business date, and it will fail if multiple dates are present in the file.
2. Upload the data files and their context files to Object Storage using the `RAP_DATA_HIST.zip` file.
3. Execute the `HIST_ZIP_FILE_LOAD_ADHOC` process.
4. Execute the `SEED_CSV_W_RTL_PO_ONORD_IT_LC_DY_F_PROCESS_ADHOC` batch process to load the data. Example Postman message body:

```
{
  "cycleName":"Adhoc", "flowName":"Adhoc",
```

```
  "processName":"SEED_CSV_W_RTL_PO_ONORD_IT_LC_DY_F_PROCESS_ADHOC"
}
```

Purchase Order data functions similar to inventory in that it is a positional fact interface and cannot be loaded or reloaded multiple times for the same dates. If you load some data for a given business date and need to change or erase it, you must truncate the tables using support utilities and then load the new data. You also cannot load the data out of order. Once you load data for one date, you may only load new files for future dates after that point. The data warehouse target tables for this data are `W_RTL_PO_DETAILS_D` (for the `ORER_HEAD.csv` file) and `W_RTL_PO_ONORD_IT_LC_DY_F` (for the `ORDER_DETAIL.csv` file).

Prior to starting nightly loads of PO data, you must also choose your configuration option for the parameter `PO_FULL_LOAD_IND` in the `C_ODI_PARAM_VW` configuration table in Manage System Configurations. By default, this parameter is set to `N`, which means that the nightly interface expects to get all delta/incremental updates to your existing PO data each night. This delta includes zero balance records for when a PO is received in the source and moves to zero units on order. If tracking and sending deltas is not possible, you may change this parameter to `Y`, which indicates that your nightly file will be a full snapshot of open order records instead. The system will automatically zero out any purchase order lines that are not included in your nightly file, which allows you to extract only the non-zero lines in your source data.

You must also choose the configuration option for the parameter `PDS_EXPORT_DAILY_ONORD`, which determines whether the `EOW_DATE` used in the export data is allowed to contain non-end-of-week (EOW) dates, or if the system must convert it to a week-ending date in all cases. When set to a value of `Y`, it means daily dates are allowed in the `EOW_DATE` field on the export, if there is a daily date in the `OTB_EOW_DATE` column of `ORDER_HEAD.csv`. When set to a value of `N`, it means the system automatically converts the input dates from `ORDER_HEAD.csv` to be week-ending dates in all cases. For a base implementation of MFP with no customizations, you may want this setting to be `N` to force the export dates to be EOW dates even if the input file has non-EOW dates from the source. If you are altering RPAS to have a different base calendar intersection, then you may want to change this to `Y` instead to allow daily dates.

Other History Loads

While sales and inventory are the most common facts to load history for, you may also want to load history for other areas such as receipts and transfers. Separate ad hoc history load processes are available for the following fact areas:

- HIST_CSV_ADJUSTMENTS_LOAD_ADHOC
- HIST_CSV_INVRECEIPTS_LOAD_ADHOC
- HIST_CSV_MARKDOWN_LOAD_ADHOC
- HIST_CSV_INVRTV_LOAD_ADHOC
- HIST_CSV_TRANSFER_LOAD_ADHOC
- HIST_CSV_DEAL_INCOME_LOAD_ADHOC
- HIST_CSV_ICMARGIN_LOAD_ADHOC
- HIST_CSV_INVRECLASS_LOAD_ADHOC
- HIST_STG_CSV_SALES_WF_LOAD_ADHOC
- HIST_SALES_WF_LOAD_ADHOC

All of these interfaces deal with transactional data (not positional) so you may use them at any time to load history files in each area.

Note:

These processes are intended to support history data for downstream applications such as AI Foundation and Planning, so the tables populated by each process by default should satisfy the data needs of those applications. Jobs not needed by those apps are not included in these processes.

Some data files used by AIF and Planning applications do not have a history load process, because the data is only used from the current business date forwards. For Purchase Order data (`ORDER_DETAIL.csv`), refer to the section below on Seed Positional Facts if you need to load the file before starting your nightly batch processing. For other areas like transfers/allocations used by Inventory Planning Optimization, those jobs are only included in the nightly batch schedule and do not require any history to be loaded.

Modifying Staged Data

If you find problems in the data you've staged in the RAP database (specific to RI/AIF input interfaces) you have the option to directly update those tables from APEX, thus allowing you to reprocess the records without uploading new files through FTS. You have the privileges to insert, delete, or update records in tables where data is staged before being loaded into the core data model, such as `W_RTL_INV_IT_LC_DY_FTS` for inventory data.

Directly updating the staging table data can be useful for quickly debugging load failures and correcting minor issues. For example, you are attempting to load `PRODUCT.csv` for the first time and you discover some required fields are missing data for some rows. You may directly update the `W_PRODUCT_DTS` table to put values in those fields and rerun the POM job, allowing you to progress with your dataload and find any additional issues before generating a new file. Similarly, you may have loaded an inventory receipts file, but discovered after staging the file that data was written to the wrong column (`INVRC_QTY` contains the `AMT` values and vice versa). You can update the fields and continue to load it to the target tables to verify it, and then correct your source data from the next run forwards only.

These privileges extend only to staging tables, such as table names ending in `FTS`, `DTS`, `FS`, or `DS`. You cannot modify internal tables holding the final fact or dimension data. You cannot modify configuration tables as they must be updated from the Control & Tactical Center. The privileges do not apply to objects in the RDX or PDS database schemas.

Reloading Dimensions

It is common to reload dimensions at various points throughout the history load, or even in-sync with every history batch run. Ensure that your core dimensions, such as the product and location hierarchies, are up-to-date and aligned with the historical data being processed. To reload dimensions, you may follow the same process as described in the Initial Dimension Load steps, ensuring that the current business load date in the system is on or before the date in history when the dimensions will be required. For example, if you are loading history files in a monthly cadence, ensure that new product and location data required for the next month has been loaded no later than the first day of that month, so it is effective for all dates in the history data files.

It is also very important to understand that history load procedures are unable to handle reclassifications that have occurred in source systems when you are loading history files. For example, if you are using current dimension files from the source system to process historical data, and the customer has reclassified products so they are no longer correct for the historical time periods, then your next history load may place sales or inventory under the new classifications, not the ones that were relevant in history. For this reason, reclassifications should be avoided if at all possible during history load activities, unless you can maintain historical dimension snapshots that will accurately reflect historical data needs.

Seed Positional Facts

Once sales and inventory history have been processed, you will need to perform seeding of the positional facts you wish to use. Seeding a fact means to load a full snapshot of the data for all active item/locations, thus establishing a baseline position for every possible record before nightly batches start loading incremental updates to those values. Seeding of positional facts should only occur once history data is complete and daily batch processing is ready to begin. Seed loads should also be done for a week-ending date, so that you do not have a partial week of daily data in the system when you start daily batches.

Instead of doing separate seed loads, you also have the option of just providing full snapshots of all positional data in your first nightly batch run. This will make the first nightly batch take a long time to complete (potentially 8+ hours) but it allows you to skip all of the steps documented below. This method of seeding the positional facts is generally the preferred approach for implementers, but if you want to perform manual seeding as a separate activity, review the rest of this section. If you are also implementing RMFCS, you can leverage Retail Data Extractor (RDE) programs for the initial seed load as part of your first nightly batch run, following the steps in that chapter instead.

If you did not previously disable the optional inventory features in `C_ODI_PARAM` (parameters `RI_INVAGE_REQ_IND`, `RA_CLR_LEVEL`, `RI_PRES_STOCK_IND`, `RI_BOH_SEEDING_IND`, `RI_MOVE_TO_CLR_IND`, and `RI_MULTI_CURRENCY_IND`) then you should review these settings now and set all parameters to `N` if the functionality is not required. Once this is done, follow the steps below to perform positional seeding:

1. Create the files containing your initial full snapshots of positional data. It may be one or more of the following:
   - `PRICE.csv`
   - `COST.csv` (used for both BCOST and NCOST data interfaces)
   - `INVENTORY.csv`
   - `ORDER_DETAIL.csv` (`ORDER_HEAD.csv` may already be loaded using dimension process, but it can also be done now, as it is required to load this file)
   - `W_RTL_INVU_IT_LC_DY_FS.dat`
2. Upload the files to Object Storage using the `RAP_DATA_HIST.zip` file.
3. Execute the `LOAD_CURRENT_BUSINESS_DATE_ADHOC` process to set the load date to be the next week-ending date after the final date in your history load.

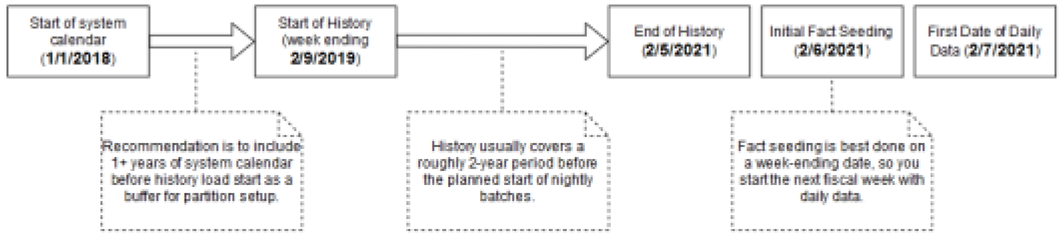
```
{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"LOAD_CURRENT_BUSINESS_DATE_ADHOC",
"requestParameters":"jobParams.ETL_BUSINESS_DATE_JOB=2017-12-31"
}
```

4. Execute the ad hoc seeding batch processes depending on which files have been provided. Sample Postman messages:

```
{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"SEED_CSV_W_RTL_PRICE_IT_LC_DY_F_PROCESS_ADHOC"
}

{
"cycleName": "Adhoc",
"flowName":"Adhoc",
"processName":"SEED_CSV_W_RTL_NCOST_IT_LC_DY_F_PROCESS_ADHOC"
}

{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"SEED_CSV_W_RTL_BCOST_IT_LC_DY_F_PROCESS_ADHOC"
}

{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"SEED_CSV_W_RTL_INV_IT_LC_DY_F_PROCESS_ADHOC"
}

{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"SEED_CSV_W_RTL_INVU_IT_LC_DY_F_PROCESS_ADHOC"
}

{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"SEED_CSV_W_RTL_PO_ONORD_IT_LC_DY_F_PROCESS_ADHOC"
}
```

Once all initial seeding is complete and data has been validated, you are ready to perform a regular batch run. Provide the data files expected for a full batch, such as `RAP_DATA.zip` or `RI_RMS_DATA.zip` for foundation data, `RI_MFP_DATA.zip` for externally-sourced planning data (for RI reporting and AI Foundation forecasting), and any AI Foundation Cloud Services files using the `ORASE_WEEKLY.zip` files. If you are sourcing daily data from RMFCS then you need to ensure that the RDE batch flow is configured to run nightly along with the RAP batch schedule. Batch dependencies between RDE and RI should be checked and enabled, if they are not already turned on.

From this point on, the nightly batch takes care of advancing the business date and loading all files, assuming that you want the first load of nightly data to occur the day after seeding. The following diagram summarizes a potential set of dates and activities using the history and seeding steps described in this chapter:



> **Note:**
>
> The sequential nature of this flow of events must be followed for positional facts (for example, inventory) but not for transactional facts (such as sales). Transactional data supports posting for dates other than what the current system date is, so you can choose to load sales history at any point in this process.

Run Nightly Batches

As soon as initial seeding is performed (or instead of initial seeding), you need to start nightly batch runs. If you are using the nightly batch to seed positional facts, ensure your first ZIP file upload for the batch has those full snapshots included. Once those full snapshots are loaded through seeding or the first full batch, then you can send incremental files rather than full snapshots.

Nightly batch schedules can be configured in parallel with the history load processes using a combination of the Customer Modules Management (in Retail Home) and the POM administration screens. It is not recommended to configure the nightly jobs manually in POM, as there are over 500 batch programs; choosing which to enable can be a time-intensive and error-prone activity. Customer Modules Management greatly simplifies this process and preserves dependencies and required process flows. Batch Orchestration describes the batch orchestration process and how you can configure batch schedules for nightly execution.

Once you move to nightly batches, you may also want to switch dimension interfaces from Full to Incremental loading of data. Several interfaces, such as the Product dimension, can be loaded incrementally, sending only the changed records every day instead of a full snapshot. These options use the `IS_INCREMENTAL` flag in the `C_ODI_PARAM_VW` table and can be accessed from the Control & Tactical Center. If you are unsure of which flags you want to change, refer to the *Retail Insights Implementation Guide* for detailed descriptions of all parameters.

> **Note:**
>
> At this time, incremental product and location loads are supported when using RDE for integration or when using legacy DAT files. CSV files should be provided as full snapshots.

As part of nightly batch uploads, also ensure that the parameter file `RA_SRC_CURR_PARAM_G.dat` is included in each ZIP package, and that it is being automatically updated with the current business date for that set of files. This file is used for business date validation so incorrect files are not processed. This file will help Oracle Support identify the current business date of a particular set of files if they need to intervene in the batch run or retrieve files from the archives for past dates. Refer to the System Parameters File section for file format details.

In summary, here are the main steps that must be completed to move from history loads to nightly batches:

1. All files must be bundled into a supported ZIP package like `RAP_DATA.zip` for the nightly uploads, and this process should be automated to occur every night.
2. Include the system parameter file `RA_SRC_CURR_PARAM_G.dat` in each nightly upload ZIP and automate the setting of the `vdate` parameter in that file (not applicable if RDE jobs are used).
3. Sync POM schedules with the Customer Module configuration using the **Sync with MDF** button in the Batch Administration screen, restart the POM schedules to reflect the changes, and then review the enabled/disabled jobs to ensure the necessary data will be processed in the batch.
4. Move the data warehouse ETL business date up to the date one day before the current nightly load (using `LOAD_CURRENT_BUSINESS_DATE_ADHOC`). The nightly load takes care of advancing the date from this point forward.
5. Close and re-open the batch schedules in POM as needed to align the POM business date with the date used in the data (all POM schedules should be open for the current business date before running the nightly batch).
6. Schedule the start time from the **Scheduler Administration** screen > **RI schedule** > **Nightly** tab. Enable it and set a start time. Restart your schedule again to pick up the new start time.

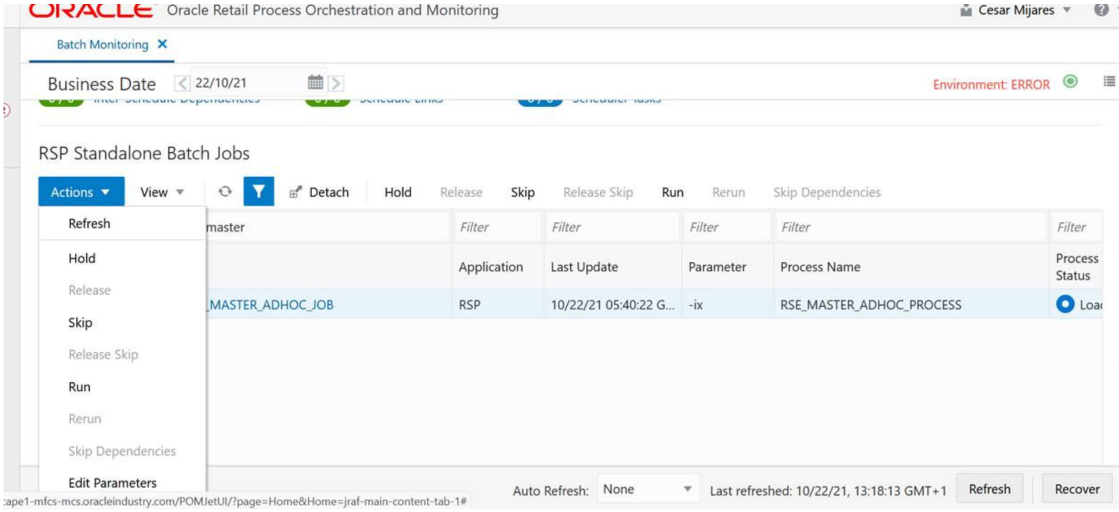
Sending Data to AI Foundation

All AI Foundation modules leverage a common batch infrastructure to initialize the core dataset, followed by ad hoc, application-specific programs to generate additional data as needed. Before loading any data into an AI Foundation module, it is necessary to complete initial dimension loads into the data warehouse using AIF DATA jobs and validate that core structures (calendar, products, locations) match what you expect to see. Once you are comfortable with the data that has been loaded in, leverage the following jobs to move data into one or more AI Foundation applications.

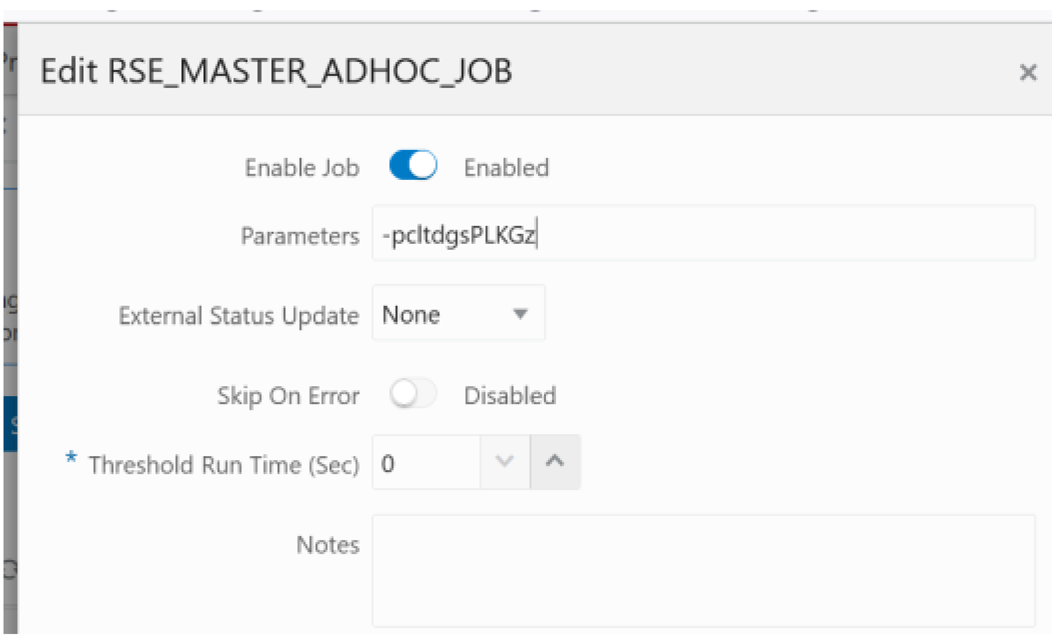
Table 3-6 Extracts for AI Foundation

POM Process Name	Usage Details
INPUT_FILES_ADHOC_PROCESS	Receive inbound zip files intended for AI Foundation, archive and extract the files. This process looks for the `ORASE_WEEKLY_ADHOC.zip` file.
RSE_MASTER_ADHOC_PROCESS	Foundation data movement from the data warehouse to AI Foundation applications, including core hierarchies and dimensional data. Accepts many different parameters to run specific steps in the load process.
<app>_MASTER_ADHOC_PROCESS	Each AI Foundation module, such as SPO or IPO, has a master job for extracting and loading data that is required for that application, in addition to the `RSE_MASTER` processes. AI Foundation module jobs may look for a combination of data from RI and input files in `ORASE_WEEKLY_ADHOC.zip`.

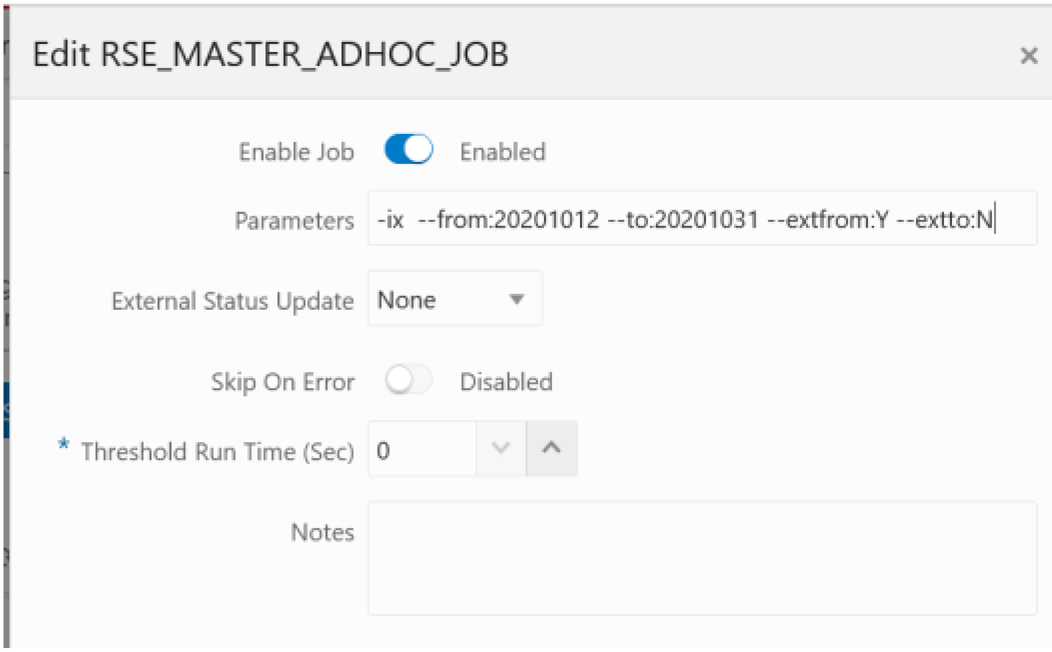
Because AI Foundation Cloud Services ad hoc procedures have been exposed using only one job in POM, they are not triggered like AIF DATA procedures. AI Foundation programs accept a number of single-character codes representing different steps in the data loading process. These codes can be provided directly in POM by editing the Parameters of the job in the Batch Monitoring screen, then executing the job through the user interface.



For example, this string of parameters will move all dimension data from the data warehouse to AI Foundation:



Additional parameters are available when moving periods of historical data, such as inventory and sales:



A typical workflow for moving core foundation data into AI Foundation is:

1. Load the core foundation files (like Calendar, Product, and Organization) using the AIF DATA schedule jobs.
2. Use the `RSE_MASTER_ADHOC_PROCESS` to move those same datasets to AI Foundation Apps, providing specific flag values to only run the needed steps. The standard set of first-time flags are `-pldg`, which loads the core hierarchies required by all the applications.
3. Load some ofyour history files for Sales using the AIF DATA jobs to validate the inputs. For example, load 1-3 months of sales for all products or a year of sales for only one department.
4. Load the same range of sales to AI Foundation using the sales steps in the master process with optional from/to date parameters. The flags to use for this are `-xwa`, which loads the transaction table `RSE_SLS_TXN` as well as all sales aggregates which are shared across AIF.
5. Repeat the previous two steps until all sales data is loaded into both RI and AI Foundation.

   Performing the process iteratively provides you early opportunities to find issues in the data before you've loaded everything, but it is not required. You can load all the data into AI Foundation at one time.

Follow the same general flow for the other application-specific, ad hoc flows into the AI Foundation modules. For a complete list of parameters in each program, refer to the AI Foundation Operations Guide.

When loading hierarchies, it is possible to have data issues on the first run due to missing or incomplete records from the customer. You may get an error like the following:

```
Error occurred in RSE_DDL_UTIL.create_local_index_stmt - Error while creating
index: CREATE UNIQUEINDEX TMP$STG$RSE_LOC_SRC_XREF_2 ON
TMP$STG$RSE_LOC_SRC_XREF
( HIER_TYPE_ID,LOC_EXT_ID,LEAF_NODE_FLG,APP_SOURCE_ID) ORA-01452: cannot
CREATE UNIQUE INDEX; duplicate keys found
```

The `INDEX` create statement tells you the name of the table and the columns that were attempting to be indexed. Querying that table is what is required to see what is duplicated or invalid in the source data. Because these tables are created dynamically when the job is run, you will need to first grant access to it using a procedure like below in IW:

```
begin RSE_SUPPORT_UTIL.grant_temp_table(p_prefix => 'STG$RSE_LOC_SRC_XREF');
end;
```

The value passed into the procedure should be everything after the `TMP$` in the table name (not the index name). The procedure also supports two other optional parameters to be used instead of or in addition to the prefix:

- `p_suffix` – Provide the ending suffix of a temporary table, usually a number like `00001101`, to grant access to all tables with that suffix
- `p_purge_flg` - Purge flag (Y/N) which indicates to drop temporary tables for a given run

Once this is executed for a given prefix, a query like this can retrieve the data causing the failure:

```
SELECT * FROM RASE01.TMP$STG$RSE_LOC_SRC_XREF WHERE
( HIER_TYPE_ID,LOC_EXT_ID,LEAF_NODE_FLG,APP_SOURCE_ID )     IN ( SELECT
        HIER_TYPE_ID,LOC_EXT_ID,LEAF_NODE_FLG,APP_SOURCE_ID          FROM
        RASE01.TMP$STG$RSE_LOC_SRC_XREF         GROUP BY
        HIER_TYPE_ID,LOC_EXT_ID,LEAF_NODE_FLG,APP_SOURCE_ID        HAVING
count(*)  > 1       )ORDER BY
HIER_TYPE_ID,LOC_EXT_ID,LEAF_NODE_FLG,APP_SOURCE_ID ;
```

For location hierarchy data the source of the issue will most commonly come from the `W_INT_ORG_DH` table. For product hierarchy, it could be `W_PROD_CAT_DH`. These are the hierarchy tables populated in the data warehouse by your foundation data loads.

Sending Data to Planning

If a Planning module is being implemented, then additional AIF DATA schedule jobs should be executed as part of the initial loads and nightly batch runs. These jobs are available through ad hoc calls, and the nightly jobs are included in the AIF DATA nightly schedule. Review the list below for more details on the core Planning extracts available.

Process Overview

Table 3-7 Extracts for Planning

POM Job Name	Usage Details
W_PDS_PRODUCT_D_JOB	Exports a full snapshot of Product master data and associated hierarchy levels, including flex fields for alternates.
W_PDS_ORGANIZATION_D_JOB	Exports a full snapshot of Location master data (for stores and virtual warehouses only) and associated hierarchy levels, including flex fields for alternates.

Table 3-7 (Cont.) Extracts for Planning

POM Job Name	Usage Details
W_PDS_CALENDAR_D_JOB	Exports a full snapshot of Calendar data at the day level and associated hierarchy levels, for both Fiscal and Gregorian calendars. **Note:** While the AIF DATA job exports the entire calendar, PDS will only import 5 years around the `RPAS_TODAY` date (current year +/- 2 years).
W_PDS_EXCH_RATE_G_JOB	Exports a full snapshot of exchange rates.
W_PDS_PRODUCT_ATTR_D_JOB	Exports a full snapshot of item-attribute relationships. This is inclusive of both diffs and UDAs.
W_PDS_DIFF_D_JOB	Exports a full snapshot of Differentiators such as Color and Size.
W_PDS_DIFF_GRP_D_JOB	Exports a full snapshot of differentiator groups (most commonly size groups used by SPO and AP).
W_PDS_UDA_D_JOB	Exports a full snapshot of User-Defined Attributes.
W_PDS_BRAND_D_JOB	Exports a full snapshot of Brand data (regardless of whether they are currently linked to any items).
W_PDS_SUPPLIER_D_JOB	Exports a full snapshot of Supplier data (regardless of whether they are currently linked to any items).
W_PDS_CUSTSEG_D_JOB	Exports a full snapshot of Customer Segments data (regardless of whether they are currently linked to any customers).
W_PDS_REPL_ATTR_IT_LC_D_JOB	Exports a full snapshot of Replenishment Item/Location Attribute data (equivalent to `REPL_ITEM_LOC` from RMFCS).
W_PDS_DEALINC_IT_LC_WK_A_JOB	Incremental extract of deal income data (transaction codes 6 and 7 from RMFCS) posted in the current business week.
W_PDS_PO_ONORD_IT_LC_WK_A_JOB	Incremental extract of future on-order amounts for the current business week, based on the expected OTB date.

Table 3-7 (Cont.) Extracts for Planning

POM Job Name	Usage Details
W_PDS_PO_ONORD_IT_LC_DY_F_JOB	Full snapshot extract of currently active purchase order details for Inventory Planning Optimization. Converts pack item data to separate order lines when provided, while also excluding the pack component lines to avoid duplicate values.
W_PDS_INV_IT_LC_WK_A_JOB	Incremental extract of inventory positions for the current business week. Inventory is always posted to the current week, there are no back-posted records. Includes pack item inventory (when available) by removing the pack component quantities that belong to pack items and inserting additional rows for the pack items.
W_PDS_SLS_IT_LC_WK_A_JOB	Incremental extract of sales transactions posted in the current business week (includes back-posted transactions to prior transaction dates).
W_PDS_INVTSF_IT_LC_WK_A_JOB	Incremental extract of inventory transfers posted in the current business week (transaction codes 30, 31, 32, 33, 37, 38 from RMFCS).
W_PDS_INVRC_IT_LC_WK_A_JOB	Incremental extract of inventory receipts posted in the current business week. Only includes purchase order receipts (transaction code 20 from RMFCS).
W_PDS_INVRTV_IT_LC_WK_A_JOB	Incremental extract of returns to vendor posted in the current business week (transaction code 24 from RMFCS).
W_PDS_INVADJ_IT_LC_WK_A_JOB	Incremental extract of inventory adjustments posted in the current business week (transaction codes 22 and 23 from RMFCS).
W_PDS_SLSWF_IT_LC_WK_A_JOB	Incremental extract of wholesale and franchise transactions posted in the current business week (transaction codes 82, 83, 84, 85, 86, 88 from RMFCS).
W_PDS_MKDN_IT_LC_WK_A_JOB	Incremental extract of markdowns posted in the current business week (transaction codes 11, 12, 13, 14, 15, 16, 17, 18 from RMFCS).
W_PDS_ALC_IT_LC_DY_F_JOB	Full snapshot extract of currently active allocations for use in Inventory Planning Optimization.
W_PDS_TSF_IT_LC_DY_F_JOB	Full snapshot extract of currently active transfers for use in Inventory Planning Optimization.
W_PDS_INV_IT_LC_WK_A_INITIAL_JOB	Initial history extract for inventory position data based on the timeframe established in C_SOURCE_CDC.

Table 3-7 (Cont.) Extracts for Planning

POM Job Name	Usage Details
W_PDS_FLEXFACT1_F_JOB	Exports flexible fact data from the data warehouse that can be at any configurable level of merchandise/location hierarchies and week level of the calendar. Can be used to extend PDS with additional measures. Flex Fact 1 is also used by LPO to display custom measures.
W_PDS_FLEXFACT2_F_JOB	Exports flexible fact data from the data warehouse that can be at any configurable level of merchandise/location hierarchies and week level of the calendar. Can be used to extend PDS with additional measures.
W_PDS_FLEXFACT3_F_JOB	Exports flexible fact data from the data warehouse that can be at any configurable level of merchandise/location hierarchies and week level of the calendar. Can be used to extend PDS with additional measures.
W_PDS_FLEXFACT4_F_JOB	Exports flexible fact data from the data warehouse that can be at any configurable level of merchandise/location hierarchies and week level of the calendar. Can be used to extend PDS with additional measures.
W_PDS_ORG_ATTR_STR_D_JOB	Exports a full snapshot of custom flex location attributes and other location attribute types to PDS, for string-type attributes only.
W_PDS_ORG_ATTR_NBR_D_JOB	Exports a full snapshot of custom flex location attributes to PDS, for number-type attributes only.
W_PDS_ORG_ATTR_DT_D_JOB	Exports a full snapshot of custom flex location attributes to PDS, for date-type attributes only.
W_PDS_PRODUCT_ATTR_STR_D_JOB	Exports a full snapshot of custom flex product attributes to PDS, for string-type attributes only.
W_PDS_PRODUCT_ATTR_NBR_D_JOB	Exports a full snapshot of custom flex product attributes to PDS, for number-type attributes only.
W_PDS_PRODUCT_ATTR_DT_D_JOB	Exports a full snapshot of custom flex product attributes to PDS, for date-type attributes only.
W_PDS_PROD_ORG_ATTR_STR_D_JOB	Exports a full snapshot of custom flex product/location attributes to PDS, for string-type attributes only.
W_PDS_PROD_ORG_ATTR_NBR_D_JOB	Exports a full snapshot of custom flex product/location attributes to PDS, for number-type attributes only.
W_PDS_PROD_ORG_ATTR_DT_D_JOB	Exports a full snapshot of custom flex product/location attributes to PDS, for date-type attributes only.

The PDS jobs are linked with several ad hoc processes in POM, providing you with the ability to extract specific datasets on-demand as you progress with history and initial data loads. The table below summarizes the ad hoc processes, which can be called using the standard methods such as cURL or Postman.

Table 3-8 AIF DATA Ad Hoc Processes for Planning

POM Process Name	Usage Details
LOAD_PDS_DIMENSION_PROCESS_ADHOC	Groups all of the dimension (D/G table) extracts for PDS into one process for ad hoc execution. Disable individual jobs in POM to run only some of them.
LOAD_PDS_FACT_PROCESS_ADHOC	Groups all of the fact (F/A table) extracts for PDS into one process for ad hoc execution. Disable individual jobs in POM to run only some of them.
LOAD_PDS_FACT_INITIAL_PROCESS_ADHOC	History extract process for inventory positions, run only to pull a full snapshot of inventory from the data warehouse for one or multiple weeks of data. You must set the start and end dates for extraction in the C_SOURCE_CDC table before running this process. The dates must be week-ending dates aligning with the first and last week to export.
C_LOAD_DATES_CLEANUP_ADHOC	Purge the job records from C_LOAD_DATES and C_LOAD_DATES_DETAILS, which is necessary when you are running the same jobs multiple times per business date.

Most of the PDS fact jobs leverage the configuration table C_SOURCE_CDC to track the data that has been extracted in each run. On the first run of an incremental job in LOAD_PDS_FACT_PROCESS_ADHOC, the job extracts all available data in a single run. From that point forwards, the extract incrementally loads only the data that has been added or modified since the last extract, based on W_UPDATE_DT columns in the source tables. There are some exceptions to this incremental process: Inventory, On Order, and Allocation/Transfer Detail interfaces. The normal incremental jobs for these interfaces will always extract the latest day's data only, because they are positional facts that send the full snapshot of current positions to PDS each time they run.

To move inventory history prior to the current day, you must use the initial inventory extract to PDS (LOAD_PDS_FACT_INITIAL_PROCESS_ADHOC). It requires manually entering the start/end dates to extract, so you must update C_SOURCE_CDC from the Control & Tactical Center for the inventory table record. LAST_MIN_DATE is the start of the history you wish to send, and LAST_MAX_DATE is the final date of history. For example, if you loaded one year of inventory, you might set LAST_MIN_DATE to 04-JUN-22 and LAST_MAX_DATE to 10-JUN-23. Make sure that the timestamps on the values entered are 00:00:00 when saved to the database, otherwise the comparison between these values and your input data may not align.

For all other jobs, the extract dates are written automatically to C_SOURCE_CDC alongside the extract table name after each execution and can be overwritten as needed when doing multiple loads or re-running for the same time period. If you run the same process more than once, use C_LOAD_DATES_CLEANUP_ADHOC to reset the run statuses before the next run, then edit

C_SOURCE_CDC to change the minimum and maximum dates that you are pulling data for. Review the table below for a summary of the configuration tables involved in PDS extracts.

Table 3-9 Planning Integration Configuration Tables

Table	Usage
C_SOURCE_CDC	Configuration and tracking table that shows the interfaces supported for data warehouse to Planning integration and the currently processed date ranges.
C_LOAD_DATES	Tracks the execution of jobs and the most recent run status of each job. Prevents running the same job repeatedly if it was already successful, unless you first erase the records from this table.
C_LOAD_DATES_DETAILS	Tracks the progress of a job's individual steps when a job supports task-level restartability. Prevents re-running of a successful step on restarting a job from POM unless you first erase the records from this table.
RAP_INTF_CFG	Configuration and tracking table for integration ETL programs between all RAP modules. Contains their most recent status, run ID, and data retention policy.
RAP_INTF_RUN_STATUS	RAP integration run history and statuses.
RAP_LOG_MSG	RAP integration logging table, specific contents will vary depending on the program and logging level.

Integration tables used to move data from the data warehouse to PDS are designed for temporary data retention and will generally maintain only the current day or week of data based on the most recent job executions. The data retention policy for every interface is found on RAP_INTF_CFG and is enforced using a weekly maintenance job called PURGE_OLD_RDX_EXPORTS_JOB. All data past its retention period will be dropped when this job runs. Accumulating excess data in integration tables would increase storage costs to the customer and potentially impact the performance of accessing these tables over time, therefore it is not advised to disable this job or change the data retention policies.

After the planning data has been extracted from the data warehouse to PDS staging tables (W_PDS_*), the Planning applications use the same programs to extract both full and incremental data for each interface. You can run the dimension and fact loads for planning from the Online Administration (OAT) tasks, or use the RPASCE schedule in POM. Refer to the relevant implementation guides for MFP, AP, or IPO for details on these processes.

Usage Examples

The following examples show how to leverage the PDS extract processes to move data from the data warehouse tables to the PDS staging tables, where the data can be picked up by the Planning applications.

Scenario 1: Initial Dimension Extract

1. Perform the initial loads into the data warehouse as described in the section Initialize Dimensions.
2. Enable all jobs in the ad hoc process LOAD_PDS_DIMENSION_PROCESS_ADHOC and execute it.
3. Verify that data has been moved successfully to the target tables, such as W_PDS_CALENDAR_D, W_PDS_PRODUCT_D, and W_PDS_ORGANIZATION_D.

Scenario 2: Initial Sales Extract

1. Perform the initial sales loads into the data warehouse as described in the section Sales History Load.
2. Enable the job `W_PDS_SLS_IT_LC_WK_A_JOB` in the ad hoc process `LOAD_PDS_FACT_PROCESS_ADHOC` and execute the process.
3. Verify that data has been moved successfully to the target tables `W_PDS_SLS_IT_LC_WK_A` and `W_PDS_GRS_SLS_IT_LC_WK_A`.
4. Repeat the same steps for any other transactional history loads, such as adjustments, transfers, and RTVs (using the appropriate PDS job as described in the prior section).

Scenario 3: Initial Inventory Extract

1. Perform the initial inventory loads into the data warehouse as described in the section Inventory Position History Load.
2. Open the `C_SOURCE_CDC` table in Manage System Configurations and locate the row for `W_RTL_INV_IT_LC_WK_A`. Edit the values in the `LAST_MIN_DATE` and `LAST_MAX_DATE` columns to fully encompass your range of historical dates in the inventory history.
3. Enable the job `W_PDS_INV_IT_LC_WK_A_INITIAL_JOB` in the ad hoc process `LOAD_PDS_FACT_INITIAL_PROCESS_ADHOC` and execute the process.
4. If the job fails with error code “ORA-01403: no data found,” it generally means that the dates in `C_SOURCE_CDC` are not set or do not align with your historical data. Update the dates and re-run the job.
5. Verify that data has been moved successfully to the target table `W_PDS_INV_IT_LC_WK_A`.

Scenario 4: Updated Data Extracts

1. Load additional history data as required for the project. For example, you may have started with one month of data for testing and are now going to load an additional 2 years of data.
2. If the `C_LOAD_DATES_CLEANUP_ADHOC` process was not run at the start of your current data load, run it now to ensure there are no previous run statuses from past PDS export runs that will interfere with new runs.
3. Run `LOAD_PDS_DIMENSION_PROCESS_ADHOC` to ensure the dimensions for PDS are in sync with the new history data being loaded.
4. Review `C_SOURCE_CDC` and alter the minimum/maximum dates on each table, if required. If you want to re-push your entire history for a fact, including the previously loaded data, then you will need to adjust the dates before running the extracts.
5. Run `LOAD_PDS_FACT_PROCESS_ADHOC` for all the facts, such as sales and receipts, that have had new history loaded in the data warehouse.
6. Run `LOAD_PDS_FACT_INITIAL_PROCESS_ADHOC` to pull the desired range of inventory periods.
7. Verify the target tables contain the expected data before starting the import into Planning applications.

Sending Daily Data to Planning

All fact interfaces from the Analytics and Planning data warehouse to the Planning applications use week-level aggregated data by default. Some interfaces support changing the data level from weekly to daily, which may be required for specific customizations or extensions that are not part of the base solutions. To change an interface to send daily data instead of weekly data, you must update the associated parameter in the C_ODI_PARAM table from the list below.

Scenario Name	Param Name	Configuration Guidance
GLOBAL	PDS_EXPORT_DAILY_SLS	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_SLS_IT_LC_WK_A and W_PDS_GRS_SLS_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_INV	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_INV_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_ONORD	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_PO_ONORD_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_SLSWF	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_SLSWF_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_INVRC	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_INVRC_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_INVTSF	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_INVTSF_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_INVADJ	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_INVADJ_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_INVRTV	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_INVRTV_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_INVRECLASS	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_INVRECLASS_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_MKDN	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_MKDN_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_DEALINC	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_DEALINC_IT_LC_WK_A.
GLOBAL	PDS_EXPORT_DAILY_ICM	Set to Y if you want daily data to be sent from the data warehouse to W_PDS_ICM_IT_LC_WK_A.

When an interface is configured for daily data, it changes the source tables used in the data warehouse. For example, weekly sales data comes from W_RTL_SLS_IT_LC_WK_A. When using daily interfaces, the sales data will come from W_RTL_SLS_IT_LC_DY_A instead. It is necessary to populate these tables as part of your historical data loads before changing any configurations in C_ODI_PARAM. The table below provides the list of daily data tables used by these configurations.

Interface Name	Source Tables
Sales	W_RTL_SLS_IT_LC_DY_A, W_RTL_SLSPK_IT_LC_DY_F
Inventory	W_RTL_INV_IT_LC_WK_A
Purchase Order	W_RTL_PO_ONORD_IT_LC_DY_F
Sales Wholesale	W_RTL_SLSWF_IT_LC_DY_F
Receipts	W_RTL_INVRC_IT_LC_DY_F
Transfers	W_RTL_INVTSF_IT_LC_DY_F
Adjustments	W_RTL_INVADJ_IT_LC_DY_F
RTVs	W_RTL_INVRTV_IT_LC_DY_F
Reclass	W_RTL_INVRECLASS_IT_LC_DY_F
Markdown	W_RTL_MKDN_IT_LC_DY_F
Deal Income	W_RTL_DEALINC_IT_LC_DY_F
IC Margin	W_RTL_ICM_IT_LC_DY_F

The inventory and purchase order (PO) interfaces use same source table for both daily and weekly extracts. Because inventory and PO data is positional, it means that every time the batch program is run, it always sends the current date's full snapshot of data. What changes between the daily and weekly configurations is the value written to `EOW_DATE`. When the interface is configured for daily data, the `EOW_DATE` will always reflect the current business date instead of only the week-ending date.

In addition to the configuration parameters in `C_ODI_PARAM`, you are also expected to change the batch jobs in the AIF DATA schedule in POM to run daily. If the jobs are set to run every day instead of only on week-ending dates, they will incrementally send only the new daily data that has been added or modified since the previous day's run, with the daily dates written on `EOW_DATE` instead of the week-ending dates. It is assumed that you have already altered your RPAS configuration to use a base intersection of day level; otherwise, having daily values on `EOW_DATE` will result in failures when running the MFP or AP batch processes.

Another difference between daily and weekly extracts is the logic used to calculate currency columns. In the weekly extracts, the amounts are pre-converted to primary currency on the source tables and then extracted as-is. For the daily extracts, the day level tables in the data warehouse do not have pre-converted values on them, so the currency conversion is done at runtime by taking the local currency amount (for example, `SLS_AMT_LCL`) and dividing by the exchange rate (`LOC_EXCHANGE_RATE`). This conversion should result in the same value as you would find on the week-level table columns (for example, `SLS_AMT`). Aside from the currency conversion, the base calculations used are the same across the daily and weekly extracts.

Customized Planning Integrations

Some implementations require additional custom data exports into Planning applications which supplement or entirely replace the standard integrations available between AIF and RPAS applications. The PL/SQL package `RAF_INTF_SUPPORT_UTIL` is available in Innovation Workbench (IW) to support this type of integration requirement. The process to publish data from IW to any existing interface will be similar to the example described here. For this example, the interface will be `RDF_FCST_PARAM_CAL_EXP`.

To review the interface configuration details for this interface, the following query can be executed:

```
SELECT * FROM rap_intf_cfg WHERE intf_name = 'RDF_FCST_PARM_CAL_EXP';
```

The `PUBLISH_APP_CODE` and `CUSTOMER_PUBLISHABLE_FLG` values are important to note from the result of this query. First, to publish data to an interface, the publishable flag must be `Y` for any interface that IW will be populating. If there is a need to publish data for an interface not configured for custom extensions, an SR is required that requests that the following update be made, which will allow publishing data for that interface:

```
UPDATE rap_intf_cfg
   SET customer_publishable_flg='Y'
 WHERE intf_name = 'RDF_FCST_PARM_CAL_EXP';
/
COMMIT;
/
```

Once the above has been done, the implementer can write code in IW that will publish data to this interface, so that the subscribing application modules can consume it. A sample of how to do this follows:

```
DECLARE
   --The following is used to capture the ID for a published interface dataset
   v_run_id    NUMBER;
   -- The following is used to allow isolated writing to the interface
   v_partition_name    VARCHAR2(30);

BEGIN
   rap_intf_support_util.prep_intf_run('PDS', 'RDF_FCST_PARM_CAL_EXP',
v_run_id, v_partition_name);

   -- Use of Append ensures efficiency, and requires PARTITION
( partition_name )
-- syntax, to ensure the ability to concurrently write to the table
   EXECUTE IMMEDIATE 'INSERT /*+ append */ INTO RDF_FCST_PARM_CAL_EXP
partition ( '||v_partition_name|| ') ....  ';

   --NOTE: the .... is to be replaced with the actual list of columns and an
appropriate SELECT statement to provide the values to insert into the table.
   --Of importance, it should be noted that the run_id column must be
populated with the value that was returned into the v_run_id variable above.
   --After the data has been populated into the above table, it can be made
available for retrieval by consuming application modules by the following:
   --Note, the 'PDS' value that is shown below, is fixed according to what
was obtained from the query of RAP_INTF_CFG.

   rap_intf_support_util.publish_data('PDS', v_run_id,
'RDF_FCST_PARM_CAL_EXP', 'Ready');

END;
/
```

Once the above steps have been completed, the data will be ready to be consumed by the application modules that use this interface.

> **Note:**
>
> If an attempt is made to call any of the routines inside `RAP_INTF_SUPPORT_UTIL` for an interface where the`CUSTOMER_PUBLISHABLE_FLG` is not set to `Y`, then an error will be provided indicating that "No Interfaces for [<values they provided to the procedure>] are allowed/available to be published from custom code." If this occurs, follow the instructions described above to submit an SR asking for permission to publish data to that interface.

Generating Forecasts for MFP

Before you can complete an MFP implementation, you must set up and run forecasting within AI Foundation Cloud Services. Review the steps below for the initial setup of Forecasting:

1. In **Manage Forecast Configurations** in the AI Foundation UI, start by setting up a run type in the Setup train stop.
2. Click the **+** icon above the table and fill in the fields in the popup. For MFP forecasting, the forecast method should be selected as **Automatic Exponential Smoothing**.
3. Create a run type for each forecast measure/forecast intersection combination that is required for MFP.
4. Create test runs in the Test train stop once you are done setting up the run types:
   a. Click a run type in the top table, then click on the **+** icon in the bottom table to create a run.
   b. If necessary, change the configurations parameters for the estimation process and forecast process in their respective tabs in the **Create Run** popup.

      For example, if you want to test a run using Bayesian method, edit the **Estimation Method** parameter in the **Estimation** tab using the edit icon above the table.
   c. After modifying and reviewing the configuration parameters, click the **Submit** button to start the run.
5. Once the run is complete, the status changes to **Forecast Generation Complete**.

Doing test runs is an optional step. In addition to that, you will need to modify and review the configurations of the run type, activate the run type, enable auto-approve and map the run type to the downstream application (in this case to MFP). In the Manage train stop, select a row, click **Edit Configurations Parameters** and edit the estimation and forecast parameters as needed. Once you are done, go to **Review** tab, click **Validate**, then close the tab.

> **Note:**
>
> If the run type is active, you will only be able to view the parameters. To edit the parameters, the run type must be inactive.

To activate the run type and enable the auto-approve, select a run type in the table and click the corresponding buttons above the table. Lastly, to map the run type to MFP, go to the Map train stop and click the **+** icon to create a new mapping.

When configuring forecasts for the MFP base implementation, the following list of forecast runs may be required, and you will want to configure and test each run type following the general workflow above. Additional runs can be added to satisfy your MFP implementation requirements.

Note:

The “Channel” level in MFP is often referred to as “Area” level in RI and AI Foundation, so be sure to select the correct levels which align to your hierarchy.

MFP Plan	MFP Levels	Method	Data Source	Measure
MFP Merch Target	Department-Channel-Week	Auto ES	Store Sales	Regular and Promotion Gross Sales Amt
MFP Merch Target	Department-Channel-Week	Auto ES	Store Sales	Clearance Gross Sales Amt
MFP Merch Target	Department-Channel-Week	Auto ES	Store Sales	Regular and Promotion Gross Sales Unit
MFP Merch Target	Department-Channel-Week	Auto ES	Store Sales	Clearance Gross Sales Unit
MFP Merch Target	Department-Channel-Week	Auto ES	Store Sales	Total Returns Amount
MFP Merch Target	Department-Channel-Week	Auto ES	Store Sales	Total Returns Units
MFP Merch Plan	Subclass-Channel-Week	Auto ES	Store Sales	Regular and Promotion Gross Sales Amt
MFP Merch Plan	Subclass-Channel-Week	Auto ES	Store Sales	Clearance Gross Sales Amt
MFP Merch Plan	Subclass-Channel-Week	Auto ES	Store Sales	Regular and Promotion Gross Sales Unit
MFP Merch Plan	Subclass-Channel-Week	Auto ES	Store Sales	Clearance Gross Sales Unit
MFP Merch Plan	Subclass-Channel-Week	Auto ES	Store Sales	Total Returns Amount
MFP Merch Plan	Subclass-Channel-Week	Auto ES	Store Sales	Total Returns Units
MFP Location Target	Company-Location-Week	Auto ES	Store Sales	Total Gross Sales Amount
MFP Location Target	Company-Location-Week	Auto ES	Store Sales	Total Gross Sales Unit
MFP Location Target	Company-Location-Week	Auto ES	Store Sales	Total Returns Amount
MFP Location Target	Company-Location-Week	Auto ES	Store Sales	Total Returns Units
MFP Location Plan	Department-Location-Week	Auto ES	Store Sales	Total Gross Sales Amount

MFP Plan	MFP Levels	Method	Data Source	Measure
MFP Location Plan	Department-Location-Week	Auto ES	Store Sales	Total Gross Sales Unit
MFP Location Plan	Department-Location-Week	Auto ES	Store Sales	Total Returns Amount
MFP Location Plan	Department-Location-Week	Auto ES	Store Sales	Total Returns Units

Generating Forecasts for Inventory Planning Optimization Cloud Service-Demand Forecasting

The same forecasting interface described in the previous section for MFP is also used to generate the base demand and initial forecasts for Inventory Planning Optimization Cloud Service-Demand Forecasting (IPOCS-Demand Forecasting). Demand and forecasts must be generated in AI Foundation as part of your Forecasting implementation. The general workflow is the same, but the forecasting levels and methods used will vary depending on your implementation needs. For example, your IPOCS-Demand Forecasting forecasts may be at an item/location/week level of granularity instead of higher levels like MFP requires. You will also use other forecasting methods such as Causal-Short Life Cycle instead of the MFP default method (Auto ES).

IPOCS-Demand Forecasting directly integrates the demand and forecast parameters between AI Foundation Cloud Services and PDS tables using the RAP data exchange layer (RDX) as needed. Outputs from the forecasting engine will be written to tables prefixed with `RSE_FCST_*`. Outputs from IPOCS-Demand Forecasting back into the data exchange layer will be in tables prefixed with `RDF_APPR_FCST_*`. For more details on importing the forecasts after they are generated, refer to the *IPO Demand Forecasting Implementation Guide*.

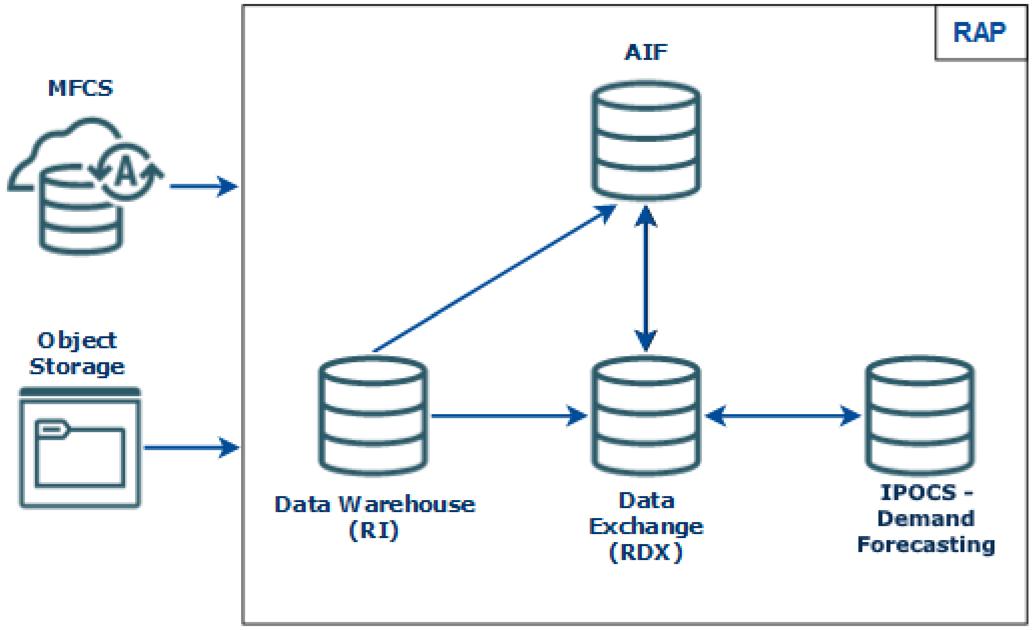
Implementation Flow Example

The steps below describe a minimal implementation workflow for IPOCS-Demand Forecasting, which has the most complex process of the Planning applications. A similar process would be followed for other Planning applications, except the integration would largely be one-way (AIF pushing data to MFP/AP). Note that these manual steps are provided for first-time implementations and testing, all jobs would be automated as part of nightly batches before going live with the application.

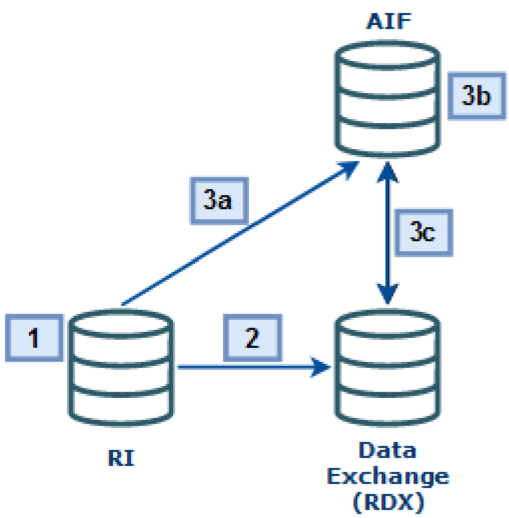
Retail Insights is used as the foundational data warehouse that collects and coordinates data on the platform. You do not need to purchase Retail Insights Cloud Service to leverage the data warehouse for storage and integration, it is included as part of any RAP solution. Regardless of which RAP solutions you are implementing, the integration flows shown below are the same.

This section has high-level steps to provide an understanding of the major events in the setup process, but detailed explanations may be found in the *AI Foundation Implementation Guide* sections on “Forecast Configuration for IPO-DF and AIF” and "Workflow for IPO-DF Implementation". Refer to that document for additional configuration guidance, this section is only a summary to help you understand the interactions between the applications.

Figure 3-2 Integration Summary



1. Integrate your foundation data (core dimensions and fact history) using either RMFCS direct loads or object storage file uploads and run the load programs following the steps described earlier in this chapter.
2. Move foundation data to the Data Exchange (RDX) using the `LOAD_PDS_DIMENSION_PROCESS_ADHOC` and `LOAD_PDS_FACT_PROCESS_ADHOC` processes.
3. AI Foundation and Forecast Setup Flow



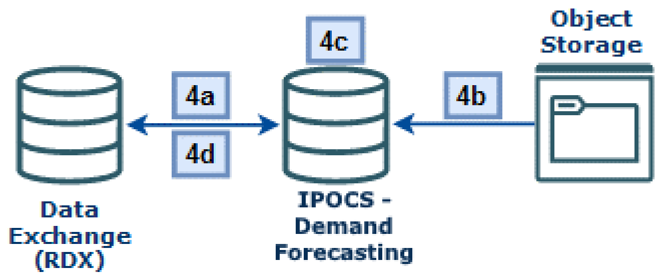
   a. Move the data to AI Foundation using the `RSE_MASTER_ADHOC_JOB` process (passing in the appropriate parameters for your data).

b. Set up run types and execute test runs in the Forecasting module of AI Foundation, then approve and map those runs to IPOCS-Demand Forecasting. Set up Flex Groups in AIF to be used with the forecasts in IPOCS-Demand Forecasting.

c. Export AIF setup data for IPOCS-Demand Forecasting to the Data Exchange (RDX) using the jobs below (MFP and AP do not require most of these jobs, instead you would simply run `RSE_FCST_EXPORT_ADHOC_PROCESS` jobs for MFP/AP exports):

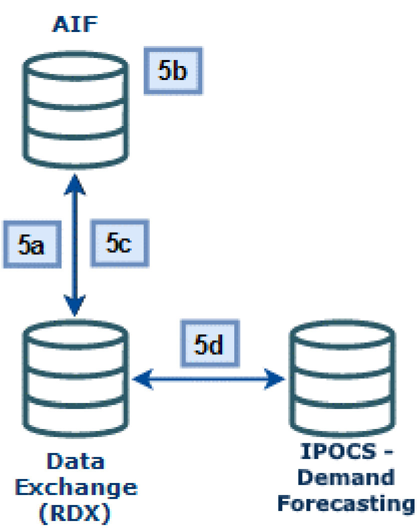
   - `RSE_FCST_RUN_TYPE_CONF_EXPORT_ADHOC_PROCESS`
   - `RSE_PROMO_OFFER_EXPORT_ADHOC_PROCESS`
   - `RSE_FCST_EXPORT_ADHOC_PROCESS` (enabling the IPOCS-Demand Forecasting job only)

4. IPOCS-Demand Forecasting Setup Flow



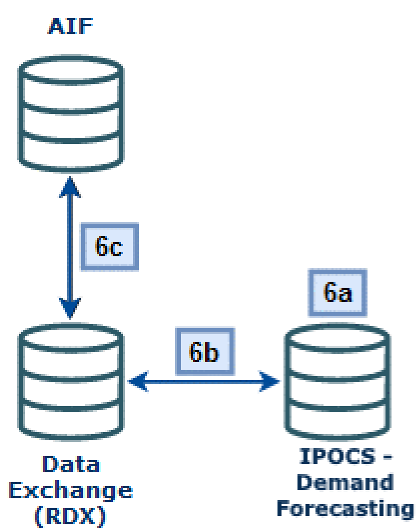
   a. Import hierarchy and foundation data from RDX to IPOCS-Demand Forecasting.
   b. Import any app-specific non-foundation files from Object Storage to IPOCS-Demand Forecasting directly.
   c. Perform your initial IPOCS-Demand Forecasting Workflow Tasks following the IPO-DF Implementation and User Guides, such as building the domain, setting up forecast parameters, new items, what-ifs, and so on.
   d. Run the IPOCS-Demand Forecasting Pre-Forecast and Export Forecast Parameters Batches.

5. Forecast Execution Flow



a. Import the updated IPOCS-Demand Forecasting parameters to AIF using the jobs:
   - `RSE_RDX_FCST_PARAM_ADHOC_PROCESS`
   - `RSE_FCST_RDX_NEW_ITEM_ENABLE_ADHOC_PROCESS`
   - `RSE_LIKE_RDX_RSE_ADHOC_PROCESS`
   - `PMO_EVENT_IND_RDF_ADHOC_PROCESS`
b. Return to the AIF Forecasting module and generate new forecasts using the IPOCS-Demand Forecasting parameters. Create new runs under the same run type as before, generate the forecast(s), approve the demand parameters, and click **Approve Base Demand and Forecast**. Ensure you activate the run type from the Manager Forecast Configurations screen and enable auto-approve (if starting nightly runs).
c. Export the forecasts using the `RSE_FCST_EXPORT_ADHOC_PROCESS` (you can directly run the IPOCS-Demand Forecasting job `RSE_RDF_FCST_EXPORT_ADHOC_JOB`), `RSE_FCST_RUN_TYPE_CONF_EXPORT_ADHOC_PROCESS`, and `RSE_PROMO_OFFER_SALES_EXPORT_ADHOC_PROCESS`.
d. Import the forecasts to IPOCS-Demand Forecasting. Also re-run any of the previous IPOCS-Demand Forecasting steps if any other data has changed since the last run.

6. Forecast Approval Flow



a. Perform IPOCS-Demand Forecasting Workflow Tasks to review/modify/approve the final forecasts (Run Post Forecast Batch, Build Forecast Review, run Forecast What-Ifs)
b. Export the approved forecast from IPOCS-Demand Forecasting using the Export Approved Forecast OAT Task or the associated POM job.
c. Use AIF POM jobs to process the approved forecast and generate flat file exports for RMFCS (if required):
   - `RSE_RDX_APPD_FCST_ADHOC_PROCESS` (import final forecast to AIF)
   - `RSE_RDF_APPR_FCST_EXPORT_ADHOC_PROCESS` (export week level)
   - `RSE_RDF_APPR_FCST_DAY_EXPORT_ADHOC_PROCESS` (export day level)

Generating Forecasts for AP

The same forecasting interface described in the previous section for MFP is also used to generate the Assortment Planning forecasts. When configuring forecasts for the AP base implementation, the following list of forecast runs may be required, and you will want to configure and test each run type. Additional runs can be added to satisfy your AP implementation requirements.

> **Note:**
>
> Although AP has an in-season plan, it still leverages Auto ES as the base forecasting method.

Bayesian (which includes plan data in the forecast) is set up as the estimation method for the run. This is also why Store Sales is set as the data source for all runs, because all runs have the ability to include plan data based on the estimation methods used (in addition to store sales).

AP Plan	AP Levels	Method	Data Source	Measure
Item Plan Pre-Season	Item-Location-Week	Auto ES	Store Sales	Regular and Promotion Gross Sales Units
Item Plan In-Season	Item-Location-Week	Auto ES (with Bayesian estimation method)	Store Sales	Regular and Promotion Gross Sales Units
Subclass Plan Pre-Season	Subclass-Location-Week	Auto ES	Store Sales	Regular and Promotion Gross Sales Units

Loading Plans to RI

If you are implementing a Planning module and need to generate plan-influenced forecasts (such as the AP in-season forecast) then you will need to first integrate your plans into RI (which acts as the singular data warehouse for this data both when the plans come from Oracle Retail solutions and when they come from outside of Oracle).

If you are pushing the plans from MFP or AP directly, then you will enable a set of POM jobs to copy plan outputs directly to RI tables. You may also use an ad hoc process to move the plan data on-demand during the implementation.

Table 3-10 Plan Extracts from MFP and AP

POM Job Name	Usage Details
W_RTL_PLAN1_PROD1_LC1_T1_FS_SDE_JOB	Extract the MFP plan output from the `MFP_PLAN1_EXP` table in the RDX schema.
W_RTL_PLAN2_PROD2_LC2_T2_FS_SDE_JOB	Extract the MFP plan output from the `MFP_PLAN2_EXP` table in the RDX schema.

Table 3-10 (Cont.) Plan Extracts from MFP and AP

POM Job Name	Usage Details
W_RTL_PLAN3_PROD3_LC3_T3_FS_SDE_JOB	Extract the MFP plan output from the `MFP_PLAN3_EXP` table in the RDX schema.
W_RTL_PLAN4_PROD4_LC4_T4_FS_SDE_JOB	Extract the MFP plan output from the `MFP_PLAN4_EXP` table in the RDX schema.
W_RTL_PLAN5_PROD5_LC5_T5_FS_SDE_JOB	Extract the AP plan output from the `AP_PLAN1_EXP` table in the RDX schema.

In all of the jobs above, the target table name is in the job name (such as `W_RTL_PLAN2_PROD2_LC2_T2_FS` as the target for the `PLAN2` extract). Once the data is moved to the staging layer in RI, the source-agnostic fact load jobs will import the data (`W_RTL_PLAN2_PROD2_LC2_T2_FS` is loaded to `W_RTL_PLAN2_PROD2_LC2_T2_F`, and so on). The fact tables in RI are then available for AI Foundation jobs to import them as needed for forecasting usage.

The plan tables in RI have configurable levels based on your Planning implementation. The default levels are aligned to the standard outputs of MFP and AP if you do not customize or extend them. If you have modified the planning solution to operate at different levels, then you must also reconfigure the RI interfaces to match. This includes the use of flexible alternate hierarchy levels (for example, the `PRODUCT_ALT.csv` interface) which require custom configuration changes to the `PLAN` interfaces before you can bring that data back to RI and AIF. These configurations are in `C_ODI_PARAM_VW`, accessed from the Control & Tactical Center in AI Foundation. For complete details on the plan configurations in RI, refer to the *Retail Insights Implementation Guide*.

Loading Forecasts to RI

If you are implementing IPOCS-Demand Forecasting and RI, then you may want to integrate your approved forecast with RI for reporting. The same RI interfaces are used both for IPOCS-Demand Forecasting forecasts and external, non-Oracle forecasts.

If you are pushing the forecast from IPOCS-Demand Forecasting, use a set of POM jobs to copy forecast outputs directly to RI tables. You may also use an ad hoc process to move the forecast data on-demand during the implementation. The process names for the two interfaces are `LOAD_PLANFC1_DATA_ADHOC` and `LOAD_PLANFC2_DATA_ADHOC`. These processes should be scheduled to run automatically outside of batch, as this is the preferred way to integrate forecast data (because the volume can be large and you do not want it to impact the nightly batch cycle times).

You must also map the desired forecast run type to the RI application code from the AI Foundation user interface. Once this is done, the integration jobs can detect data to extract. Only one run type can be mapped to RI at this time; use a SKU/store/week run type, unless you have reconfigured the RI interfaces for some other intersection.

Table 3-11 Forecast Extracts from AIF and IPOCS-Demand Forecasting

POM Job Name	Usage Details
W_RTL_PLANFC_PROD1_LC1_T1_FS_SDE_JOB	Extracts the baseline forecast from the AI Foundation table RSE_FCST_DEMAND_DTL_CAL_EXP in the RDX schema.
W_RTL_PLANFC_PROD2_LC2_T2_FS_SDE_JOB	Extracts the final, approved forecast from the AI Foundation view RSE_RDF_APPR_FCST_VW, which contains only the approved forecast export from IPOCS-Demand Forecasting. Requires a run type mapped to the RI app code.

In all of the jobs above, the target table name is in the job name (such as W_RTL_PLANFC_PROD1_LC1_T1_FS as the target for the PLANFC1 extract). Once the data is moved to the staging layer in RI, the source-agnostic fact load jobs will import the data (W_RTL_PLANFC_PROD1_LC1_T1_FS is loaded to W_RTL_PLANFC_PROD1_LC1_T1_F, and so on).

The forecast tables in RI have configurable levels based on your implementation. The default levels are aligned to the standard outputs of AI Foundation and IPOCS-Demand Forecasting (item/location/week) if you do not customize or extend them. If you have modified the IPOCS-Demand Forecasting solution to operate at different levels, then you must also reconfigure the RI interfaces to match. These configurations are in C_ODI_PARAM_VW, accessed from the Control & Tactical Center in AI Foundation. For complete details on the forecast configurations in RI, refer to the *Retail Insights Implementation Guide*.

Loading Aggregate History Data

If you are only looking to implement a Planning solution or certain basic modules of AI Foundation and you cannot provide transaction-level history data, then you have the option to load pre-aggregated historical fact data into RAP, bypassing the usual transaction-level interfaces. The custom fact aggregates allow for up to 4 different intersections of measure data at levels at or above item/location/date. The fact columns are generic and accept various numeric measure data across all typical functional areas (sales, receipts, inventory, transfers, and so on) in the same interface. The non-numeric fields on each interface are only for integration to PDS; they won't be used by AI Foundation.

The aggregate fact interface files and their associated data warehouse tables are listed below. Refer to the *RAP Interfaces Guide* in My Oracle Support for complete file specifications.

Filename	Staging Table	Target Table
W_RTL_FACT1_PROD1_LC1_T1_FS.dat	W_RTL_FACT1_PROD1_LC1_T1_FS	W_RTL_FACT1_PROD1_LC1_T1_F
W_RTL_FACT2_PROD2_LC2_T2_FS.dat	W_RTL_FACT2_PROD2_LC2_T2_FS	W_RTL_FACT2_PROD2_LC2_T2_F
W_RTL_FACT3_PROD3_LC3_T3_FS.dat	W_RTL_FACT3_PROD3_LC3_T3_FS	W_RTL_FACT3_PROD3_LC3_T3_F
W_RTL_FACT4_PROD4_LC4_T4_FS.dat	W_RTL_FACT4_PROD4_LC4_T4_FS	W_RTL_FACT4_PROD4_LC4_T4_F

You must configure the data intersections for these tables before you can use them, as each table can only have one intersection defined. The parameters are in the `C_ODI_PARAM_VW` table in the Control & Tactical Center Manage System Configurations screen. The parameters for each interface are listed below.

Parameter Name	Default Value
RI_FACT1_ATTR_LEVEL	ALL
RI_FACT1_CAL_LEVEL	DAY
RI_FACT1_ORG_LEVEL	LOCATION
RI_FACT1_PROD_LEVEL	ITEM
RI_FACT1_SUPP_LEVEL	ALL
RI_FACT2_ATTR_LEVEL	ALL
RI_FACT2_CAL_LEVEL	DAY
RI_FACT2_ORG_LEVEL	LOCATION
RI_FACT2_PROD_LEVEL	ITEM
RI_FACT2_SUPP_LEVEL	ALL
RI_FACT3_ATTR_LEVEL	ALL
RI_FACT3_CAL_LEVEL	DAY
RI_FACT3_ORG_LEVEL	LOCATION
RI_FACT3_PROD_LEVEL	ITEM
RI_FACT3_SUPP_LEVEL	ALL
RI_FACT4_ATTR_LEVEL	ALL
RI_FACT4_CAL_LEVEL	DAY
RI_FACT4_ORG_LEVEL	LOCATION
RI_FACT4_PROD_LEVEL	ITEM
RI_FACT4_SUPP_LEVEL	ALL

In the current release, the `ATTR` and `SUPP` parameters should remain as `ALL`; other options are not supported when integrating the data throughout the platform. You can configure the `PROD`, `ORG`, and `CAL` levels for each interface to match the intersection of data being loaded there. Valid parameter values for each type are listed below.

Product (PROD)	Organization (ORG)	Calendar (CAL)
CMP	COMPANY	YEAR
DIV	CHAIN	HALFYEAR
GRP	AREA	QUARTER
DEPT	REGION	PERIOD
CLS	DISTRICT	WEEK
SBC	LOCATION	DAY
ITEM		

Before using the interfaces, you must also partition them using either day- or week-level partitioning (depending on the data intersections specified above). Partitioning is controlled using two tables accessible from the Control & Tactical Center: `C_MODULE_ARTIFACT` and `C_MODULE_EXACT_TABLE`.

In C_MODULE_ARTIFACT, locate the rows where the module code starts with FACT (such as FACT1) and set them to both ACTIVE_FLG=Y and PARTITION_FLG=Y.

Locate the same modules in C_MODULE_EXACT_TABLE and modify the columns PARTITION_COLUMN_TYPE and PARTITION_INTERVAL to be either WK (for week level data) or DY (for day level data). Partitioning will then be performed the first time the FACT programs are executed.

After the interfaces are configured, you must prepare the data files for upload following these guidelines:

- All key columns on the interface must be populated, even if you have specified ALL as the data level. You should use a default value of -1 to populate these fields. This includes the fields PROD_DH_NUM, PROD_DH_ATTR, ORG_DH_NUM, SUPPLIER_NUM, and CAL_DATE.
- The calendar (CAL_DATE) field must always be a date. If loading the data above day level, use the end-of-period date. The format must match the date mask specified on the context (CTX) file.
- The PLANNING_TYPE_CODE field was originally used to specify whether the planning domain was COST or RETAIL, but this makes no functional difference in the datafile at this time and can be set to any value for your own reference.
- The VERSION_NUM field is for future use, it can be defaulted to a value of 0.
- The DATASOURCE_NUM_ID field must be provided with a hard-coded value of 1, similar to all other interface specifications that contain this column.
- The INTEGRATION_ID field must be provided with a unique value that identifies the record, such as a concatenation of all primary key values.
- The data file should be formatted based on the options specified in the associated context (CTX) file, such as choosing to use pipes or commas for delimiters.

To load the files into the data warehouse, use the standalone process in the AIF DATA schedule named HIST_AGGR_FACT_LOAD_ADHOC. Make sure you enable and run all jobs related to your target table (such as W_RTL_FACT1_PROD1_LC1_T1_F). The sequence of jobs to be executed for one table is like this:

1. VARIABLE_REFRESH_JOB
2. ETL_REFRESH_JOB
3. W_RTL_FACT1_PROD1_LC1_T1_FS_COPY_JOB
4. W_RTL_FACT1_PROD1_LC1_T1_FS_STG_JOB
5. W_FACT1_PROD1_LC1_T1_F_VALIDATOR_JOB
6. W_RTL_FACT1_PROD1_LC1_T1_TMP_JOB
7. W_RTL_FACT1_PROD1_LC1_T1_F_JOB
8. RABE_TO_RTLWSP_GRANTS_JOB

To push the data downstream to Planning applications, use the standalone processes named LOAD_PDS_FACT1_AGGR_PROCESS_ADHOC through LOAD_PDS_FACT4_AGGR_PROCESS_ADHOC. The planning loads will populate RDX schema tables, such as W_PDS_FACT1_PROD1_LC1_T1_F, which can then be used for customizations and extensions in PDS (in the GA solutions this data would not be used).

After data is loaded into the core data warehouse tables, you will also need to configure and load the AI Foundation application tables before the data is accessible to any AIF modules. Because the intersections for the data are flexible and the populated columns are unknown until the data is loaded, you will need to instruct the system on how to use your aggregate data.

The measure metadata will be stored in the AIF table `RSE_MD_CDA`. This table is loaded programmatically using an ad hoc job in the RSP schedule named `RSE_AGGREGATE_METADATA_LOAD_ADHOC_JOB`. The program will detect the columns with data and add entries for each measure with a generic name assigned. Once the program is complete, you can modify the UI display name to be something meaningful to end-users from the Control & Tactical Center.

The measures themselves will first be loaded into `RSE_PR_LC_CAL_CDA`, which is the staging area in AIF to prepare the measures for the applications. After the metadata is configured, you may run another ad hoc job in the RSP schedule named `RSE_AGGREGATE_ACTUALS_LOAD_ADHOC_JOB`. This will populate the columns in `RSE_PR_LC_CAL_CDA` based on their metadata.

Lastly, you must map the measure data into the application tables that require access to aggregate facts. This is performed using the configuration table `RSE_MD_FACT_COLUMN_MAP`, which is accessible for inserts and updates in the Control & Tactical Center. Possible configuration options supported by the AIF applications will be listed in their respective implementation guides, but a sample set of values is provide below for a sales and inventory measure mapping, which will be the most common use cases:

SOURCE_TABLE	SOURCE_COLUMN	TARGET_TABLE	TARGET_COLUMN
W_RTL_FACT1_PROD1_LC1_T1_F	SLSRG_QTY	RSE_SLS_PR_LC_WK	SLS_QTY
W_RTL_FACT1_PROD1_LC1_T1_F	BOH_QTY	RSE_INV_PR_LC_WK_A	INV_QTY_BOH

Separate POM jobs are included in the RSP schedule to move the data from the CDA tables to their final target tables. The jobs will come in pairs and have job names ending in `AGGR_MEAS_SETUP_ADHOC_JOB` followed by `AGGR_MEAS_PROCESS_ADHOC_JOB`. For example, to load the sales table in the sample mapping, use `RSE_SLS_PR_LC_WK_AGGR_MEAS_SETUP_ADHOC_JOB` and `RSE_SLS_PR_LC_WK_AGGR_MEAS_PROCESS_ADHOC_JOB`. For additional details on the individual AIF application usage of these mappings and jobs, refer to the *AIF Implementation Guide*.

If you need in-season forecasts, then you must plan to configure MFP or AP plan exports to RI as part of your planning implementation. You must populate the same columns on the plan exports that you are using on the FACT1-4 interfaces for actuals. When doing in-season forecasts with aggregated data, it expects the same column in a `PLAN` and `FACT` table at the same intersection so that it can load the associated plan measure for the actuals and do a plan-influenced forecast run. For example, if you are populating the `SLS_QTY` column on the FACT1 interface, then you must also send an `SLS_QTY` value on the PLAN1 interface or else it won't be used in the plan-influenced forecast.

Migrate Data Between Environments

The process of moving all your data from one cloud environment to another is referred to as a "lift-and-shift" process. It is common to perform all of the dataload activities in this chapter in one environment, then have the final dataset copied into another environment (instead of reloading it all from the beginning). This activity is performed by Oracle and can be requested using a Service Request. The lift-and-shift process can take several days to complete, depending on data volumes and how many other Oracle applications may be involved. It is expected that you will raise SRs to schedule the activity at least 2-3 weeks in advance of the target date.

When requesting the activity, you may specify if you need to migrate the entire database (both data and structures) or only the data. The first time doing this process must be a full migration of data and structures to synchronize the source and target environments. It is currently recommended to begin your implementation in the Production environment to avoid needing a lift-and-shift in order to go live. Around the time of your go-live date, you can request a lift-and-shift be done into your non-production environments to synchronize the data for future use.

> **Note:**
>
> The product versions between the source and target must be aligned. It is the project team's responsibility to plan for appropriate upgrades and lift-and-shift activities such that this will be true.

4 Integration with Merchandising

This chapter describes the various integrations between Retail Merchandising Foundation Cloud Services (RMFCS) and the Retail Analytics and Planning platform. RMFCS can be used as the primary source of foundation data for RAP and pre-built integrations and batch programs exist to move data between cloud applications. You may also use an on-premise installation of the Retail Merchandising System (RMS), in which case you must establish the integration to the RAP cloud following the guidance in this chapter.

Architecture Overview

In release versions 21 and earlier, the integration between RMFCS and RI/AIF used a tool named the Retail Data Extractor (RDE) to generate data files for RI/AIF to consume. From version 23, these programs have been fully integrated to the RAP batch flow and directly insert the data from RMFCS to RAP. The integration uses an instance of Oracle Golden Gate to copy RMFCS tables to the local database, where the Data Extractor jobs can source all required data and transform it for use in RAP. If you are familiar with the prior RDE architecture, then you need to be aware of the following major changes:

1. `RDE_DM01` database objects are now located in the `RADM01` schema. `RDE_RMS01` database objects are now in the `RABE01USER` schema. The `RABE01USER` now has access to extract records from RMFCS through the Golden Gate replicated schema.
2. The `C_ODI_PARAM` configuration tables have been merged and all RDE configurations are accessed from the Control & Tactical Center.
3. File-based integration has been removed. All data is moved directly between database source and target tables with no option to produce flat files.
4. RDE's batch schedule has been merged with RI's schedule in POM. Jobs have been renamed and assigned modules such that it is easy to identify and disable/enable RDE jobs as needed.
5. Customer Engagement (CE) integration jobs have been included in RDE for when CE is set up to replicate data to RAP. File-based integration is no longer required.
6. All jobs relating to file extraction, ZIP file creation, or data cleanup in RMS have been removed.

Because RDE jobs are now a part of the RAP nightly batch cycle, they have been assigned modules in the Customer Modules framework (accessed using Retail Home) and can be enabled or disabled in bulk depending on your use-cases.

- `RDE_RMS` – These are the RDE components in relation to RMS
- `RDE_CE` – These are the RDE components in relation to CE

If you are not integrating data from RMFCS or CE then you will need to disable these modules to prevent them from running in your nightly batch cycles. RI and AIF jobs are programmed to start automatically after the RDE jobs complete, but if the RDE jobs are disabled then the dependencies will be ignored.

Merchandising Foundation Cloud Service Data Mapping

After Golden Gate makes the copy of all data residing in the source Merchandising Foundation Cloud Service (MFCS) environment, there is a mapping layer that connects the replicated database schema to Retail Data Extractor (RDE) programs. This mapping layer consists of two parts:

- The replicated schema contains wrapper view (WV) objects on top of the cloned tables to provide a layer of abstraction between the source and destination. If the underlying table changes in the source it does not necessarily have to impact the target systems, since the view could remain unchanged. For example, the source table `ITEM_MASTER` has a wrapper view `RDS_WV_ITEM_MASTER` on top of it.
- Within the target RAP database, synonyms are created to map each wrapper view back to its source system name. The synonyms are what are granted to RDE and Innovation Workbench. For example, there is a synonym named `ITEM_MASTER` which is used by RDE to extract data from the `RDS_WV_ITEM_MASTER` view. The `ITEM_MASTER` synonym can also be directly queried from Innovation Workbench instead of trying to query the original wrapper view object. The synonyms reside in the `RABE01USER` user in the database.

This architecture is used for almost all objects from MFCS, with the following exceptions:

- The synonym for `ITEM_LOC_SOH` points to the MFCS table `ITEM_LOC_SOH_EOD`, which is the end-of-day inventory snapshot. This is required for RAP to ensure it receives the same end-of-day inventory positions that MFCS and other applications will show. The `ITEM_LOC_SOH` table itself is not used in RAP integrations at this time due to the constantly changing data from 24/7 inventory program activity.
- The synonym for `MV_CURRENCY_CONVERSION_RATES` refers to `V_CURRENCY_CONVERSION_RATES`. This is because `MV_CURRENCY_CONVERSION_RATES` cannot be replicated to a Golden Gate schema, so MFCS created the `V_CURRENCY_CONVERSION_RATES` view as a replacement.
- RAP uses the `RMS_CALENDAR` synonym to refer to the `CALENDAR` table. This is because RI also has a `CALENDAR` table in its schema and the object names would have conflicted within RDE.

Batch Schedule Definitions

The RDE jobs have been labeled and categorized by their primary purpose, so you can easily identify jobs you may want to enable or disable for your implementation. The bulk of the RDE jobs are divided into two main classifications, which are dimension jobs (those with `RDE_EXTRACT_DIM_*` in their job names) and fact jobs (those with `RDE_EXTRACT_FACT_*` in their job names). They are also grouped into execution phases based on their functional usage, as described in the tables below. The phases are all run in parallel to each other and don’t have dependencies between them, they are purely for ease of use.

Table 4-1 RDE Dimension Phases

DIMENSION PHASE No.	GROUPING FACTOR
P1	Consists of dimension jobs that populate the `GRP1`, `GRP2`, and `GRP3` staging tables
P2	Consists of the Security jobs (RAF table entries)

Table 4-1 (Cont.) RDE Dimension Phases

DIMENSION PHASE No.	GROUPING FACTOR
P3	Consists of dimension jobs related to the item (for example, Merch hierarchy, item charges, item season, `RDW_DELETE*`)
P4	Consists of dimension jobs related to Custom Flex Attributes (CFA)
P5	Consists of dimension jobs related to Replenishment
P6	Consists of dimension jobs related to Merch Organization Hierarchy (for example, location list, location traits, organization financial information, and so on)
P7	Consists of dimension jobs related to Transfers (for example, transfer details, transfer charges, and so on)
P8	Consists of dimension jobs related to extract of codes used by Merchandising (for example, Inventory Adjustment reasons, Inventory status types, codes, and so on)
P9	Consists of dimension jobs related to the supplier (for example, supplier traits, supplier information, and so on)
P10	Consists of dimension jobs related to Purchase Orders (for example, PO Details, Shipment Details, ELC, and so on)
P11	Consists of dimension jobs related to allocation (for example, Allocation detail, allocation charges, and so on)
P12	Consists of dimension jobs related to Merchandising Calendar information
P13	Consists of dimension jobs related to Merchandising promotions
P14	Consists of dimension jobs related to employee information
P15	Consists of dimension jobs related to competitor information
P16	Consists of dimension jobs related to Merchandising Parameter tables (`*_GS`).
P17	Consists of dimension jobs related to buyer information
P18	Consists of Merchandising Lookup Dimension information.
P19	Consists of Company Closure Dimension information.
P20	Consists of Deal Dimension information.

Table 4-2 RDE Fact Phases

FACT PHASE No.	GROUPING FACTOR
P1	Consists of cost fact information jobs (for example, Base Cost, net Cost, and so on)
P2	Consists of price fact information jobs (for example, item price, competitor price, and so on)
P3	Consists of purchase order fact jobs
P4	Consists of transfer / RTV fact jobs
P5	Consists of Supplier fact jobs
P6	Consists of Stock Ledger fact jobs

Table 4-2 (Cont.) RDE Fact Phases

FACT PHASE No.	GROUPING FACTOR
P7	Consists of fact jobs related to inventory (for example, unavailable inventory, inventory receipt, and so on)
P8	Consists of fact jobs related to transactions - they mostly read from `IF_TRAN_DATA` (for example, deal income, intercompany margin, and so on)
P9	Consists of Allocation fact jobs
P10	Consists of stock count fact jobs
P11	Consists of replenishment fact jobs

Ad Hoc Processes

There are several standalone ad hoc processes available for executing the RDE programs outside of a normal batch cycle. These processes can be used to integrate dimension or fact data to RAP during initial implementation, or simply to run the extracts and validate the outputs without actually loading them into the platform. The table below summarizes these processes and their usage.

Table 4-3 RDE Ad Hoc Processes

Process Name	Usage
RDE_DIM_FLOW_ADHOC	Execute the dimension data extracts from RMFCS and write the result to RAP input staging tables directly. Data can then be moved into RAP if desired by using the `RI_DIM_INITIAL_ADHOC` process.
RDE_CE_DIM_FLOW_ADHOC	Execute the dimension data extracts from CE and write the result to RAP input staging tables directly. Data can then be moved into RAP if desired by using the `RI_DIM_INITIAL_ADHOC` process.
RDE_FACT_FLOW_ADHOC	Execute the fact data extracts from RMFCS and write the result to RAP input staging tables directly. This is mainly intended to allow a complete run of RDE fact jobs outside the normal batch process, loading the data into RI would require the use of numerous fact jobs and processes depending on the data needed. This process may run as full or incremental loads depending on the POM system options used for ODI runs.
RDE_POSITIONALFACT_SEED_ADHOC	Run all of the positional fact seed jobs needed to create full snapshots of positional data in RAP. After running these jobs, you may use the `SEED_*_ADHOC` processes to load each dataset (you must disable the COPY/STG steps of the seed processes before running them, because those steps will attempt to load from flat files instead of RMFCS).

Table 4-3 (Cont.) RDE Ad Hoc Processes

Process Name	Usage
RDE_INVPOS_SEED_ADHOC	Run the inventory position fact seed job needed to create full snapshots of inventory data in RI. After running these jobs, you may use the `SEED_W_RTL_INV_IT_LC_DY_F_PROCESS_ADHOC` process to load it (you must disable the COPY/STG steps of the seed processes before running them, because those steps will attempt to load from flat files instead of RMFCS).
RDE_INVRTVFACT_INITIAL_ADHOC	Extract RTV transaction history from RMFCS to RAP staging tables. Load the data into RAP using the `HIST_CSV_INVRTV_LOAD_ADHOC` process (you must disable the COPY/STG steps of the processes before running them, because those steps will attempt to load from flat files instead of RMFCS).
RDE_IVADJILDSDE_INITIAL_ADHOC	Extract adjustment transaction history from RMFCS to RAP staging tables. Load the data into RAP using the `HIST_CSV_ADJUSTMENTS_LOAD_ADHOC` process (you must disable the COPY/STG steps of the processes before running them, because those steps will attempt to load from flat files instead of RMFCS).
RDE_IVTSFILDSDE_INITIAL_ADHOC	Extract transfer transaction history from RMFCS to RAP staging tables. Load the data into RAP using the `HIST_CSV_TRANSFER_LOAD_ADHOC` process (you must disable the COPY/STG steps of the processes before running them, because those steps will attempt to load from flat files instead of RMFCS).
RDE_DEALINILDSDE_INITIAL_ADHOC	Extract deal income transaction history from RMFCS to RAP staging tables. Load the data into RAP using the `HIST_CSV_DEAL_INCOME_LOAD_ADHOC` process (you must disable the COPY/STG steps of the processes before running them, because those steps will attempt to load from flat files instead of RMFCS).
RDE_INVRECLASSSDE_HIST_ADHOC	Extract inventory reclass transaction history from RMFCS to RAP staging tables. Load the data into RAP using the `HIST_CSV_INVRECLASS_LOAD_ADHOC` process (you must disable the COPY/STG steps of the processes before running them, because those steps will attempt to load from flat files instead of RMFCS).
RDE_INTCMPMRGINSDE_HIST_ADHOC	Extract intercompany margin transaction history from RMFCS to RAP staging tables. Load the data into RAP using the `HIST_CSV_ICMARGIN_LOAD_ADHOC` process (you must disable the COPY/STG steps of the processes before running them, because those steps will attempt to load from flat files instead of RMFCS).

Table 4-3 (Cont.) RDE Ad Hoc Processes

Process Name	Usage
RDE_EXTRACT_SALES_ADHOC	Extract daily sales data from the Sales Audit staging tables to the RAP staging tables. This process is mainly intended to test the sales extracts outside the normal batch and to validate the data transformations, or use the RAP sales history load ad hoc processes to bring the sales into the platform.
RDE_TSFILDSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the transfer details fact (`W_RTL_TSF_IT_LC_DY_FS`). Intended for data validation and history conversion. Meant to be loaded into the RAP data warehouse using the nightly batch jobs.
RDE_SHIPDETAILSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the shipment details dimension (`W_RTL_SHIP_DETAILS_DS`). Intended for data validation and history conversion. Meant to be loaded into RAP data warehouse using the nightly batch jobs.
RDE_SHIPILDSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the shipment details fact (`W_RTL_SHIP_IT_LC_DY_FS`). Intended for data validation and history conversion. Meant to be loaded into RAP data warehouse using the nightly batch jobs.
RDE_ALLOCDETAILDYSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the allocation dimension (`W_RTL_ALC_DETAILS_DS`). Intended for data validation and history conversion. Meant to be loaded into RAP data warehouse using the nightly batch jobs.
RDE_POONALCILDSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the PO orders on allocation fact (`W_RTL_PO_ONALC_IT_LC_DY_FS`). Intended for data validation and history conversion. Meant to be loaded into RAP data warehouse using the nightly batch jobs.
RDE_REPLDAYSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the Replenishment Days dimension (`W_RTL_REPL_DAY_DS`). Intended for data validation and history conversion. Meant to be loaded into RAP data warehouse using the nightly batch jobs.
RDE_REPLSUPDIMSDE_INITIAL_ADHOC	Extracts a full snapshot of data from RMFCS for the Replenishment Supplier Dims dimension (`W_RTL_REPL_SUP_SIM_DS`). Intended for data validation and history conversion. Meant to be loaded into RAP data warehouse using the nightly batch jobs.

Batch Grants and Synonym Setup (Gen 2 Architecture)

Retail Data Extractor (RDE) programs require various database grants and synonyms in order to function on the Golden Gate replicated objects and move that data into RAP for further processing. Additional grants and synonyms are also used to expose data to Innovation Workbench (IW) for implementers to directly query. These configurations are applied during the provisioning and application patching; however, in rare instances, it is possible for a grant or synonym to become invalidated outside of those times. Symptoms of this issue are:

- RDE batch programs fail with an error like ORA-00942: table or view does not exist, and the program was previously working without error
- A SQL query from APEX returns an error like ORA-00942: table or view does not exist, and you are sure the object in question should be available to query

If you ever need to re-apply these grants and synonyms, ad hoc programs are available in POM for this purpose. The way to use these processes is to run the GRANT process of a particular type first, followed by the SYNONYM process of the same type.

Table 4-4 RDE Synonym and Grant Processes

Process Name	Usage
RDE_GRANT_MFCS_TO_BATCH_ADHOC	Ad hoc process that grants the MFCS Replicating Views to batch user (`RABE01USER`).
RDE_GRANT_MFCS_TO_APEX_ADHOC	Ad hoc process that grants the MFCS Replicating Views to Retail Workspace (IW).
RDE_CREATE_SYNONYM_FOR_MFCS_ADHOC	Ad hoc process that creates synonyms for the MFCS Replicating Views to batch user (`RABE01USER`).
RDE_CREATE_SYNONYM_FOR_MFCS_APEX_ADHOC	Ad hoc process that creates synonyms for the MFCS Replicating Views to Retail Workspace (IW).
RDE_GRANT_ORCE_TO_BATCH_ADHOC	Ad hoc process that grants the ORCE Replicating Views to batch user (`RABE01USER`).
RDE_GRANT_ORCE_TO_APEX_ADHOC	Ad hoc process that grants the ORCE Replicating Views to Retail Workspace (IW).
RDE_CREATE_SYNONYM_FOR_ORCE_ADHOC	Ad hoc process that creates synonyms for the ORCE Replicating Views to batch user (`RABE01USER`).
RDE_CREATE_SYNONYM_FOR_ORCE_APEX_ADHOC	Ad hoc process that creates synonyms for the ORCE Replicating Views to Retail Workspace (IW).
RDE_GRANT_BATCH_TO_APEX_ADHOC	Ad hoc process that grants batch-owned RDE temporary tables and views to Retail Workspace (IW).
RDE_CREATE_SYNONYM_FOR_BATCH_APEX_ADHOC	Ad hoc process that creates synonyms for the batch-owned RDE temporary tables and views to Retail Workspace (IW).

Batch Dependency Setup (Gen 2 Architecture)

RDE jobs have pre-defined dependencies with RI, as well as interschedule dependencies with RMFCS. When you enable the RDE jobs, the dependencies with RI/AIF will be enabled automatically, but you will need to manually enable/disable the RMFCS interschedule dependencies based on your needs.

You should start with all dependencies enabled, and only disable them if you are trying to run the batch cycle out of sync from the RMFCS batch. The inter-schedule dependencies fall into two categories: discreet jobs that perform some check on RMFCS data, and POM dependencies that cross-reference another RMFCS batch program. The first category of jobs check the availability of data from the RMFCS signaling table called `RMS_RDE_BATCH_STATUS`. The RDE jobs that check the signaling table in RMFCS are:

- `RDE_INTERSCHED_CHECK_RESAEXTRACT_PROCESS` / `RDE_INTERSCHED_CHECK_RESAEXTRACT_JOB` - Checks the completion of the `RESA_EXTRACT` job in RMFCS
- `RDE_INTERSCHED_CHECK_INVSNAPSHOT_PROCESS` / `RDE_INTERSCHED_CHECK_INVSNAPSHOT_JOB` - Checks the completion of the `INVENTORY_SNAPSHOT` job that signifies that the `ITEM_LOC_SOH_EOD` table in RMFCS is now available for the RDE extract
- `RDE_INTERSCHED_CHECK_STAGETRANDATA_PROCESS` / `RDE_INTERSCHED_CHECK_STAGETRANDATA_JOB` - Checks the completion of the `STAGE_TRAN_DATA` job that signifies whether the `IF_TRAN_DATA` table in RMFCS is now available for the RDE extract

If the RDE jobs run in parallel with the RMFCS batch, then all these jobs must be enabled. If you are running RDE jobs outside the RMFCS batch, then these jobs must be disabled during those runs. The jobs will wait indefinitely for a signal from the RMFCS batch, which they will never receive if you are running RDE jobs independently.

The second category of dependencies are found on the RDE jobs themselves when you click on a job to view its details in POM or click the **Interschedule Dependencies** link in Batch Monitoring UI. These jobs are listed below, along with the RMFCS jobs they depend on. You must verify these are enabled before trying to run RDE batches (unless the associated RMFCS job is disabled, in which case the RDE dependency can be turned off as well). If any of these are disabled, you will need to use Batch Administration to enable them by locating each job and clicking into the details to enable all dependencies for it.

- `RDE_SETUP_INCRMNTL_DEALACT_PROCESS` / `RDE_SETUP_INCRMNTL_DEALACT_JOB` – This RDE job waits for the following MFCS jobs to complete:
  - `RPM_PRICE_EVENT_EXECUTION_PROCESS` / `RPM_PRICE_EVENT_EXECUTION_JOB`
- `RDE_EXTRACT_DIM_P5_REPLDAYSDE_PROCESS` / `RDE_EXTRACT_DIM_P5_REPLDAYSDE_JOB` – This RDE job waits for the following MFCS jobs to complete:
  - `REPLENISHMENT_PROCESS` / `RPLEXT_JOB`
- `RDE_EXTRACT_DIM_P3_PRDITMATTRSDE_PROCESS` / `RDE_EXTRACT_DIM_P3_PRDITMATTRSDE_JOB` – This RDE job waits for the following MFCS jobs to complete:
  - `REPLENISHMENT_PROCESS` / `RPLEXT_JOB`
- `CSTISLDSDE_PROCESS` / `CSTISLDSDE_JOB` – This RDE job waits for the following MFCS jobs to complete:
  - ALLOCBT_PROCESS / ALLOCBT_JOB
  - BATCH_RFMCURRCONV_PROCESS / BATCH_RFMCURRCONV_JOB
  - COSTCOMPUPD_ELCEXPRG_PROCESS / ELCEXCPRG_JOB
  - EDIDLCON_PROCESS / EDIDLCON_JOB
  - EXPORT_STG_PURGE_PROCESS / EXPORT_STG_PURGE_JOB
  - EDIUPAVL_PROCESS / EDIUPAVL_JOB
  - LIKESTOREBATCH_PROCESS / LIKESTOREBATCH_JOB
  - POSCDNLD_PROCESS / POSCDNLD_POST_JOB
  - REPLINDBATCH_PROCESS / REPLINDBATCH_JOB
  - SALESPROCESS_PROCESS / SALESUPLOADARCH_JOB

  - STKVAR_PROCESS / STKVAR_JOB
- `RDEBATCH_INITIAL_START_PROCESS` / `RDEBATCH_INITIAL_START_MILEMARKER_JOB` – This RDE job waits for the RMFCS job `STOP_RIB_ADAPTOR_INV_PROCESS` / `STOP_RIB_ADAPTOR_INV_JOB` to complete.

If you cannot see any dependencies in the POM UI, then your POM system options may have them disabled. Make sure to check the System Configuration for AIF DATA and ensure the dependency options are set to Enabled.

External Dependencies	**Enabled**
InterSchedule Dependencies	**Enabled**

Batch Link Setup (Gen 2 Architecture)

Aside from the dependencies, you also need to be aware of the Schedule Link that is defined between MERCH and AIF DATA schedules. By default, the schedule link will be disabled, so you must go into the AIF DATA schedule links section from Batch Monitoring and enable any links shown. The schedule link will allow RDE and RI jobs to start automatically after Merchandising jobs are run in their nightly batch cycle. There is one schedule link defined that will trigger the RDE job `RDEBATCH_INITIAL_START_MILEMARKER_JOB` after Merchandising runs. If the MERCH and AIF DATA schedules are in the same POM instance, then you must enable the Schedule Link before running any batches. If they are in different instances, then the link will not be visible and cannot be used.

Module Setup in Retail Home (Gen 2 Architecture)

Before you configure any individual jobs in POM itself, it is best to set up your Customer Modules in Retail Home to reflect your planned data flows. Go to the Customer Modules Management screen as an administrator user and review the options following this guidance:

- Under the **RAP > RAP_COMMON** section:
  - Enable or disable the **BATCH** modules per your functional needs (enable everything if unsure).
  - Disable the **HISTORY** module if you are planning to immediately start loading Merchandising data. Enable it if you will load history data using CSV file interfaces.
  - Enable or disable the AIF modules based on your AI Foundation solution plans. Ensure **AIF > CONTROLFILES** is disabled as RDE jobs supercede this functionality.
  - Enable the **RDE_CE** and **RDE_RMS** modules fully, depending on which source applications are available in your environments.
  - Disable **RDXBATCH** if you are not using any Planning solution in RAP or enable it if you do plan to use MFP, AP, or IPO.
  - Disable **SIBATCH** and **SICONTROLFILES** module sets, because they are superceded by RDE integration.
  - Disable the **ZIP_FILES** modules unless you have a need for one of them for non-Mechandising data.
- (For RI customers) Under the **RI > RCI** section, enable or disable the module depending on your plans to use Retail Insights customer-related functionality. Disable the **RI > RCI > CONTROLFILES** and **SICONTROLFILES** modules, because RDE jobs replace this

functionality. The **BATCH** modules can be left enabled if you are unsure whether these modules are needed.

- (For RI customers) Under the **RI > RMI** section, enable or disable the module depending on your plans to use Retail Insights merchandising-related functionality. Disable the **RI > RMI > CONTROLFILES** and **SICONTROLFILES** modules, because RDE jobs replace this functionality. The **BATCH** modules can be left enabled if you are unsure whether these modules are needed.

Once all modules are configured, go back into the POM Batch Administration UI and perform a **Sync with MDF** action on the AIF DATA schedule. This is a one-time activity to streamline the POM schedule setup, after which you will want to perform a review of the POM schedule and refine the enabled/disabled jobs further to cover any specific file or data requirements.

Even if you are not syncing with MDF at this time, you must still perform the Retail Home setup because the CONTROLFILES and SICONTROLFILES modules are used implicitly by AIF DATA schedule programs to know which data files to expect in the batch runs. Any misconfiguration could lead to the program `DAT_FILE_VALIDATE_JOB` failing or running for several hours while waiting for data files you didn't provide. Similarly, if you misconfigure the ZIP_FILES modules, then you will encounter errors/delays in the `ZIP_FILE_WAIT_JOB` as it looks for the expected ZIP files.

Batch Job Setup (Gen 2 Architecture)

The way you configure the integration with RMFCS varies depending on where your Merchandising applications reside. In this scenario, you have both RMFCS and CE in our 2nd generation architecture and the version number is 22.1 or greater.

If you followed the Retail Home setup steps prior to this section, then most of the jobs listed below should already be configured to your specifications. However, it is still good to validate the necessary jobs are enabled/disabled per the requirements before attempting any batch run. Take the following steps to review/configure the RDE portion of the AIF DATA batch schedule:

- Enable the jobs `RDE_SETUP_INCRMNTL_RESA_JOB`, `RDE_SETUP_INCRMNTL_DEALACT_JOB`, `RDE_SETUP_INCRMNTL_JOB`, and `RDE_INCRMNTL_AUDIT_PRG_JOB` which are mandatory for sales integration with Sales Audit and Deals integration with MFCS.
- Disable the batch schedule link between `BATCH_INITIAL_START_PROCESS / GENERIC_BATCH_MILE_MARKER_JOB` and RDE schedule's `RDE_BATCHFINAL_PROCESS / RDE_BATCHFINAL_EXTLOAD_SUCCESS_JOB` (if one is visible). This batch link should be disabled now because RDE is a part of AIF DATA's schedule and is on the same POM and batch pod, which was not true in prior architectures.
- If the client opts not to integrate the ORCE customer data, the ORCE jobs can be disabled (those with `RDE_EXTRACT_CE_*`). These jobs are under the following modules:
  - RDE_CE
  - RDE_CE_BATCH
  - RDE_CE_REQUIRED
  - RDE_CE_OPTIONAL
  - RDE_CE_CUSTOMER
  - RDE_CE_CUSTSEG
  - RDE_CE_LOYALTY

- If the client opts to integrate the ORCE customer data (which means that the ORCE jobs are enabled - those with `RDE_EXTRACT_CE_*`), the following AIF DATA jobs should be disabled as the customer data will directly be populated without an input file:
  - W_RTL_CUST_DEDUP_DS_COPY_JOB
  - W_RTL_CUST_DEDUP_DS_STG_JOB
  - W_RTL_CUST_LYL_AWD_TRX_DY_FS_COPY_JOB
  - W_RTL_CUST_LYL_AWD_TRX_DY_FS_STG_JOB
  - W_RTL_CUST_LYL_TRX_LC_DY_FS_STG_JOB
  - W_RTL_CUST_LYL_TRX_LC_DY_FS_COPY_JOB
  - W_RTL_CUST_LYL_ACCT_DS_COPY_JOB
  - W_RTL_CUST_LYL_PROG_DS_COPY_JOB
  - W_RTL_CUSTSEG_DS_COPY_JOB
  - W_RTL_CUSTSEG_DS_STG_JOB
  - W_RTL_CUSTSEG_DS_ORASE_JOB
  - W_RTL_CUST_CUSTSEG_DS_COPY_JOB
  - W_RTL_CUST_CUSTSEG_DS_STG_JOB
  - W_RTL_CUST_CUSTSEG_DS_ORASE_JOB
  - W_RTL_CUSTSEG_ATTR_DS_COPY_JOB
  - W_RTL_CUSTSEG_ATTR_DS_STG_JOB
  - W_RTL_CUST_HOUSEHOLD_DS_COPY_JOB
  - W_RTL_CUST_HOUSEHOLD_DS_STG_JOB
  - W_RTL_CUST_ADDRESS_DS_COPY_JOB
  - W_RTL_CUST_ADDRESS_DS_STG_JOB
  - W_PARTY_PER_DS_COPY_JOB
  - W_PARTY_PER_DS_STG_JOB
  - W_RTL_PARTY_PER_ATTR_DS_COPY_JOB
  - W_RTL_PARTY_PER_ATTR_DS_STG_JOB
  - W_HOUSEHOLD_DS_COPY_JOB
  - W_HOUSEHOLD_DS_STG_JOB
- Disable most of the AIF DATA copy jobs (those with `*_COPY_JOB`) except ones needed for non-RMFCS sources. These jobs should be disabled because these jobs will copy files and upload them from object storage. This is not needed because data is loaded directly into the staging tables and flat files are not expected to arrive for processing. Most of these jobs are under the following modules:
  - RI_DAT_STAGE
  - RSP_DAT_STAGE
- Disable most of the AIF DATA stage jobs (those with `*_STG_JOB`)) except ones needed for non-RMFCS sources. These jobs should be disabled as these jobs read from a flat file which are not available if using this integration. Most of these jobs are under the following modules:

  - RI_DAT_STAGE
  - RSP_DAT_STAGE
- Disable the RAP Simplified Interface jobs (those with `SI_*`, `COPY_SI_*`, and `STG_SI_*` at the start of the name) as RMFCS will be the source of data to feed RI. Most of these jobs are under the modules with the patterns below:
  - RI_SI*
  - RSP_SI*
- Disable the AIF DATA program `W_PROD_CAT_DH_CLOSE_JOB`, which is used to close unused hierarchy levels when non-Merchandising incremental hierarchy loads are used in RAP. It must not run with Merchandising, because the product hierarchy data is already being managed by RDE extracts.
- Disable the AIF DATA programs `ETL_REFRESH_JOB` and `BATCH_START_NOTIFICATION_JOB` (specifically the versions belonging to process `SIL_INITIAL_PROCESS`) because these are redundant with jobs included in the RDE schedule.
- If you are not providing any flat file uploads and using only RMFCS data, you may disable the jobs in `CONTROL_FILE_VALIDATION_PROCESS`, which will prevent any data files from being processed (and potentially overwriting the RMFCS data).
- Disable the job named `TRUNCATE_STAGE_TABLES_JOB`, which is used only for data file loads and cannot be run when RDE jobs are used for direct integration. A similar job named `RDE_TRUNCATE_STAGE_TABLES_JOB` should remain enabled as this does apply to RDE job execution.

Batch Job Setup (Gen 1 Architecture)

If your RMFCS version is in the Oracle cloud but the version number is 19.3 or earlier, then follow this process to configure the integration. In this case, RDE is a separate module installed in the RMFCS cloud, so the batch must be reconfigured accordingly to have the correct dependencies.

1. Disable all the inter-schedule dependencies related to RDE in the RI schedule, as there is a separate batch schedule used for RDE that contains these:

- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `ALLOCBT_PROCESS / ALLOCBT_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `BATCH_RFMCURRCONV_PROCESS / BATCH_RFMCURRCONV_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `COSTCOMPUPD_ELCEXPRG_PROCESS / ELCEXCPRG_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `EDIDLCON_PROCESS / EDIDLCON_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `EXPORT_STG_PURGE_PROCESS / EXPORT_STG_PURGE_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `EDIUPAVL_PROCESS / EDIUPAVL_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `LIKESTOREBATCH_PROCESS / LIKESTOREBATCH_JOB`
- `CSTISLDSDE_PROCESS / CSTISLDSDE_JOB` dependency with RMFCS `POSCDNLD_PROCESS / POSCDNLD_POST_JOB`

- CSTISLDSDE_PROCESS / CSTISLDSDE_JOB dependency with RMFCS REPLINDBATCH_PROCESS / REPLINDBATCH_JOB
- CSTISLDSDE_PROCESS / CSTISLDSDE_JOB dependency with RMFCS SALESPROCESS_PROCESS / SALESUPLOADARCH_JOB
- CSTISLDSDE_PROCESS / CSTISLDSDE_JOB dependency with RMFCS STKVAR_PROCESS / STKVAR_JOB
- RDEBATCH_INITIAL_START_PROCESS / RDEBATCH_INITIAL_START_MILEMARKER_JOB dependency with RMFCS STOP_RIB_ADAPTOR_INV_PROCESS / STOP_RIB_ADAPTOR_INV_JOB

1. Disable the batch link related to RMFCS in the RI schedule:
   a. SETUP_PROCESS / REFRESHODIVARIABLES_JOB dependency with RMFCS STOP_RIB_ADAPTOR_INV_PROCESS / STOP_RIB_ADAPTOR_INV_JOB
2. Disable all the ORCE jobs in AIF DATA (those with RDE_EXTRACT_CE*). These jobs should not be executed because RDE is not in Gen 2 Architecture. The Customer Data jobs in RI should be enabled instead (for example, W_RTL_CUSTSEG_DS_COPY_JOB, W_RTL_CUSTSEG_DS_STG_JOB)
3. Disable all the RDE jobs in AIF DATA (those with RDE_EXTRACT_*). These jobs should not be executed as a separate RDE job schedule in POM will be setup for it. These jobs are under the RDE_RMS_* modules.
4. Make sure that the AIF DATA copy jobs (those with *_COPY_JOB) are enabled. These jobs should be enabled as these jobs will copy files and upload them to the object storage which is the source of the files for RI processing.
5. Make sure that the AIF DATA stage jobs (those with *_STG_JOB) are enabled. These jobs should be enabled as these jobs will read from flat files and load them into the RI staging tables.
6. Disable the AIF DATA Simplified Interface jobs (those with SI_*) as MFCS will be the source of data to feed RI. Most of these jobs are under the following modules with the pattern below:
   a. RI_SI*
   b. RSP_SI*

Batch Setup for RMS On-Premise

If your RMS application is installed on a local server outside the Oracle cloud, then you will need to integrate your local RDE installation with the RAP cloud following the guidance below. Additionally, you should also perform all the steps in the prior section to disable the RDE components of the RI POM schedule and enable the file load procedures, since you will be sending in files from RDE.

1. Download the latest RDE version 22 patch from My Oracle Support, as the changes to support Object Storage upload are deployed by running the installer and upgrading your RDE environment.
2. Disable FTP Configuration in RDE by setting the input.do.install.ftp to false in the ant.install.properties file. This must be disabled because File Transfer Services (FTS) for the Retail Analytics and Planning cloud services are made available in this release, replacing the current SFTP process
3. Check that the FTS configuration file ra_objstore.cfg is available in RDE's $MMHOME/etc directory. The FTS configuration file contains the following variable set-up used for the Object Storage:

- RA_FTS_OBJSTORE_IND – This will be set to Y so that FTS will be enabled
- RA_FTS_OBJSTORE_URL – This is the Base URL
- RA_FTS_OBJSTORE_ENVNAMESPACE – This is the Tenant
- RA_FTS_OBJSTORE_IDCS_URL – This is the IDCS URL appended with /oauth2/v1/token at the end
- RA_FTS_OBJSTORE_IDCS_CLIENTID – This is the Client ID
- RA_FTS_OBJSTORE_IDCS_CLIENTSECRET – This is the Client ID Secret
- RA_FTS_OBJSTORE_IDCS_SCOPE – This is the IDCS Scope
- RI_OBJSTORE_UPLOAD_PREFIX – This is the Storage Prefix and is set to ris/incomingpointing to the correct Object Storage directory for RI input files

Refer to the File Transfer Services section of this document for instructions on how to get the values for each of the variables above.

4. Enable the File Transfer Service (FTS) in RDE by setting the RA_FTS_OBJSTORE_IND to Y in the FTS Configuration file ra_objstore.cfg found in RDE's $MMHOME/etc directory. This must be enabled so that the RDE nightly zip file job (RTLRDEZIP_PROCESS / RTLRDEZIP_JOB) and all existing ad hoc zip file jobs (RTLUASDE_INITIAL_DIMMENSION_LOAD_ADHOC / RTLRDEZIP_HIST_JOB, RTLRDEZIP_HIST_PROCESS_ADHOC / RTLRDEZIP_HIST_JOB, INVRTVFACT_ADHOC / ADHOCINVRTVSDE_JOB, SEEDPOSITIONALFACT_ADHOC / SEEDRDEZIP_JOB) will automatically upload files to the Object Storage through FTS for RI to pick up and download for further processing.
5. Once these changes are applied, it will no longer be possible to upload to SFTP; you will be sending the ZIP files only to Object Storage as specified in the install properties and configuration changes.

RDE Job Configuration

RDE programs have many configuration options available in the C_ODI_PARAM_VW table in the Control & Tactical Center. These must be reviewed prior to running RDE jobs. For any RDE job having an incremental flag below, the associated jobs in the RI/AIF DATA batch schedule also have incremental flags that must be updated at the same time. Having the RDE and AIF DATA flags be out of sync can result in data loss.

In addition to these parameters, it is also important to understand the MFCS settings relating to data purging and configure them accordingly before using RDE to extract data. Because RDE runs only at the end of the day after MFCS has started a nightly cycle, it is critical that no table in MFCS is set to purge data immediately (purge days <= 1). It is best if all purge-related MFCS settings are extended far enough so that, even if an RDE batch is missed for a day or two, there will not be any data lost due to MFCS nightly purge programs.

One MFCS setting in particular, EDI_REV_DAYS, must be updated to be high enough so that there is no chance of purging old order revisions while they might still receive updates that RDE needs to extract. We recommend changing this value to be at least 30 days (or longer, if you have any processes that might update an order 30+ days after initially closing it). The associated purchase order purge setting, ORDER_HISTORY_MONTHS, must also be more than a month out.

Scenario	Parameter	Usage
GLOBAL	RPM_PROMO_EVENT_LEVEL	Enable (set to Y) if using legacy RPM on-premise functionality.

Scenario	Parameter	Usage
GLOBAL	CE_PROMO_BRAND_TARGET	Determines whether promotion details from Customer Engagement (CE) will use the `BRAND` or `SUPPLIER_SITE` field. Promotions from CE have an optional brand identifier but it may contain either a brand ID or a supplier ID, so you will need to configure which of these two columns in `W_RTL_PROMO_CE_IT_LC_DS` will be used. Default = BRAND
GLOBAL	CUST_DEDUP_TYPE_CODE	Determines which customer type in Customer Engagement will be used to import deduplicated customer IDs for the purpose of updating the `W_PARTY_PER_D` table. Default = PREVIOUS_RELATE_ID
GLOBAL	RETURN_REASON_CAT_CODE	Merchandising code type for customer return reason codes.
GLOBAL	RTVR_REASON_CAT_CODE	Merchandising code type for RTV reason codes.
GLOBAL	WHOLESALE_CHANNEL	Identify whether there is a wholesale channel setup in Merchandising.
GLOBAL	ITEM_GRP1_IS_INCREMENTAL	Controls whether item attributes are incremental or full snapshots on the daily load.
GLOBAL	RMS_VERS_CHECK	Controls the behavior of code that is linked to a specific Merchandising version.
GLOBAL	RA_INV_WAC_IND	Controls the inventory cost calculation in RDE. When set to `Y` it will use Weighted Average Cost (WAC) as the item cost for all items. When set to `N` it will dynamically load Merchandising valuation methods set per department or item and apply them, choosing from average cost, unit cost, and retail-based cost.
GLOBAL	RA_INV_TAX_IND	Controls the calculation and removal of tax amounts from retail valuation of stock on hand and on-order amounts. When set to `N`, only simple VAT (SVAT) calculations are supported and non-VAT items are left as-is. When set to `Y`, the system dynamically loads Merchandising global tax and VAT information and applies it by item/location.

Scenario	Parameter	Usage
GLOBAL	RA_SLS_TAX_IND	Controls whether the sales will extract retail and discount amount exclusive of VAT. If value is Y then the retail and discount amounts in the sales extract will be VAT-exclusive. If value is N, then the retail and discount amounts in the sales extract will be VAT-inclusive. Value defaults to N as that was the default program behavior in earlier versions, before this configuration was added.
GLOBAL	RDE_INCRMNTL_AU_PRG_DAYS	This parameter is used to determine the retention period of the C_RDE_INCRMNTL_TBL_CTRL_AU integration audit table, set as a number in days.
SDE_RETAILINVRECEIPTSFACT	VWH_NO_ALC_RCPTS	Specify a type of warehouse that cannot receive allocations in the feed to RI. Converts the allocations to normal transfer receipts. Uses codes from VWH_TYPE column in Merchandising.
SDE_RETAILINVRECEIPTSFACT	STORE_NO_ALC_RCPTS	Specify a type of store that cannot receive allocations in the feed to RI. Converts the allocations to normal transfer receipts.
SDE_RETAILITEMDIMENSION	IS_INCREMENTAL	Controls whether the product dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILITEMLOCATIONRANGEDIMENSION	IS_INCREMENTAL	Controls whether the product location range dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILITEMSUPPLIERDIMENSION	IS_INCREMENTAL	Controls whether the supplier-item dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILSUBSTITUTEITEMDIMENSION	IS_INCREMENTAL	Controls whether the substitute-item dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILITEMLOCATIONDIMENSION	IS_INCREMENTAL	Controls whether the product location attribute dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILITEMLOCCFADIMENSION	IS_INCREMENTAL	Controls whether the product location CFAS dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILSUPPCFADIMENSION	IS_INCREMENTAL	Controls whether the supplier CFAS dimension is a full snapshot or incremental changes only for the daily load.

Scenario	Parameter	Usage
SDE_RETAILLOCATIONCFADIMENSION	IS_INCREMENTAL	Controls whether the location CFAS dimension is a full snapshot or incremental changes only for the daily load.
SDE_RETAILITEMCFADIMENSION	IS_INCREMENTAL	Controls whether the product CFAS dimension is a full snapshot or incremental changes only for the daily load.

The flags that impact the incremental/full load behavior for the AIF DATA jobs linked to RDE jobs are also provided below and should be configured at the same time.

Scenario	Parameter	Usage
SIL_ITEMDIMENSION	IS_INCREMENTAL	Controls whether the product dimension is a full snapshot or incremental changes only for the daily load.
SIL_RETAILITEMCFADIMENSION	IS_INCREMENTAL	Controls whether the product CFAS dimension is a full snapshot or incremental changes only for the daily load.
SIL_RETAILITEMLOCCFADIMENSION	IS_INCREMENTAL	Controls whether the product location CFAS dimension is a full snapshot or incremental changes only for the daily load.
SIL_RETAILLOCATIONCFADIMENSION	IS_INCREMENTAL	Controls whether the location CFAS dimension is a full snapshot or incremental changes only for the daily load.
SIL_RETAILSUPPCFADIMENSION	IS_INCREMENTAL	Controls whether the supplier CFAS dimension is a full snapshot or incremental changes only for the daily load.
SIL_RETAILSUBSTITUTEITEMDIMENSION	IS_INCREMENTAL	Controls whether the substitute item dimension is a full snapshot or incremental changes only for the daily load.
SIL_RETAILPROMOTIONDIMENSION	RI_INCREMENTAL_IND	Controls whether the promotion dimension is a full snapshot or incremental changes only for the daily load.

Using RDE for Calendar Setup (Gen 2 Architecture)

RDE can be used to integrate the calendar from Merchandising for initial setup of a new RAP environment; however there are several manual steps you must take to perform this activity. You have the option of following the steps below, or you can perform the file-based calendar load as defined in Calendar Setup. Whichever option you choose, the calendar must be set up before you can load any other data from Merchandising.

1. The out-of-box Merchandising calendar does not conform to the preferred RAP calendar structure. The RAP calendar must start from Day 1 of a fiscal year, and the Merchandising calendar does not (if no changes were made to it after provisioning). You must use the

Calendar Maintenance functions of Merchandising to edit the business calendar, adding or deleting periods such that the first fiscal period defined is period 1 of the fiscal year, and the first date for that period is day 1 of that fiscal year. Refer to the Configure Calendar section of the *Merchandising Implementation Guide* for details.. For example, if your fiscal calendar starts in February 2020, then the first record in the `CALENDAR` table must be for this same month. `YEAR_454` will be `2020` and `MONTH_454` will be `2`.

2. Configure `C_ODI_PARAM` for your `START_DT` and `END_DT`. `START_DT` must be set in one of these ways:
   a. The first day of a fiscal year in the calendar (that is, the same date as the `FIRST_DAY` value for the first fiscal period of an FY).
   b. Some date earlier than anything in the `CALENDAR` table. If your first `CALENDAR` period is February 2020, then `START_DT` can be set to `20190101`. The only reason to do this is if you want the Gregorian calendar to have additional Gregorian years available for any downstream use.
3. Configure the jobs in the AIF DATA schedule for loading the data. You will need the RDE job `RDE_EXTRACT_DIM_P12_MCALPERIODSDE_JOB` to extract the data, which is part of the `RDE_DIM_FLOW_ADHOC` process. Configure the `CALENDAR_LOAD_ADHOC` process to load this data and perform initial environment setup activities. You must disable the following jobs:
   - `COPY_SI_CALENDAR_JOB`
   - `STG_SI_CALENDAR_JOB`
   - `SI_W_MCAL_PERIODS_DS_JOB`
4. Run `RDE_EXTRACT_DIM_P12_MCALPERIODSDE_JOB` to extract the data from Merchandising and verify that the table `W_MCAL_PERIOD_DS` contains the full calendar information by querying it from APEX.
5. Run the modified `CALENDAR_LOAD_ADHOC` process and all later steps starting with step 4 in the Calendar Setup.
6. If you have any failure on the `DIM_CALENDAR_VALIDATOR_JOB`, it means your Merchandising calendar may need more changes to align with the RAP data requirements. Modify the data in the source and start over, being sure to use `C_LOAD_DATES_CLEANUP_ADHOC` to clear any failure statuses from the database before restarting.

If the validator job rule `CAL_R2` continues to happen after updating the Merchandising calendar, but you know that your first calendar year is not meeting the requirements and want to bypass the error for now, then you would need to update the table `C_DIM_RULE_LIST` from the Control & Tactical Center. Change the error type for this rule to `W` so that it only throws a warning in the batch program instead of failing. For example, if your Merchandising calendar starts in 2010 and you are not going to load any data until 2020 in RAP, then having an invalid first year of the calendar is not going to block you from other data load activities.

Using RDE for Dimension Loads (Gen 2 Architecture)

RDE can be used to integrate the foundation dimensions such as Product and Location hierarchies from Merchandising for initial setup of a new RAP environment. However, there are several manual steps you must take to perform this activity.

- The AIF DATA ad hoc process `RDE_DIM_FLOW_ADHOC` must be configured and run. Enable all jobs in the process if you want to extract all foundation dimensions, or selectively disable jobs you do not wish to run. Restart your POM schedule and then execute the process. The process will extract data from Merchandising tables (replicated to RAP through Golden Gate Hub) and directly insert data to RAP foundation staging tables. For example, the job `RDE_EXTRACT_DIM_P3_PRDHIERSDE_JOB` will source data from Merchandising hierarchy tables like `CLASS` and `DEPS` and directly insert them into the `W_PROD_CAT_DHS` table in RAP.

- Some dimension extracts support full or incremental extract options as parameters instead of database configurations. The jobs that support this will have parameters on the job such as `rdeitemsupcountrydimext.ksh #SysOpt.rdeMMHOME I`, where the last value is either `I` or `F` for incremental/full extraction logic. You have the option to switch between `I` and `F` as needed to change the extract logic. `RDE_EXTRACT_CLRPRICESDE_XTERN_JOB` and `RDE_EXTRACT_DIM_P5_REPLSUPDIMSDE_JOB` are two examples of jobs that support this parameter.
- The AIF DATA ad hoc process `RI_DIM_INITIAL_ADHOC` will then be used to import the foundation data from the staging tables to the target tables. Before running the process, you must make sure all jobs are disabled relating to flat file loads, because you are bypassing those steps with the RDE jobs. Disable all jobs having a name with `COPY`, `STG`, `STAGING`, or `SI` in the job name. Following the same flow for product hierarchy data as in step 1, you need to disable these jobs relating to the product hierarchy flat file load:
  - COPY_SI_PRODUCT_JOB
  - STG_SI_PRODUCT_JOB
  - SI_W_PROD_CAT_DHS_JOB

The remaining job that is active for the product hierarchy load is `W_PROD_CAT_DH_TYPE1_JOB`, which loads the RDE data from `W_PROD_CAT_DHS` to the `W_PROD_CAT_DH` table.

- Once you have disabled all flat file jobs, restart the POM schedule as needed and then run the `RI_DIM_INITIAL_ADHOC` process. This moves all Merchandising data from the RAP staging tables into the data warehouse internal tables. Once all jobs are complete, the remaining steps to move data to AIF and PDS are the same as documented in the Data Loads and Initial Batch Processing chapter.

Using RDE for Initial Seeding (Gen 2 Architecture)

RDE nightly batch programs can be used to perform initial seeding and full snapshots of positional facts without running any ad hoc processes for data loading. This provides a way to seamlessly transition from history data loads to nightly batch loads.

Prerequisites for starting this process are:

- You must have already initialized the RAP calendar (either by using RDE as described in the prior sections or by loading a calendar file following the Calendar Setup process).
- If you will load any historical data for prior dates, you must do that first and come back to this section when you are ready to cut over from history loads to RDE direct integration.
- The current business date in the data warehouse must be one day prior to the current business date being extracted from MFCS (also known as `VDATE`). If this was not already updated as part of loading historical data, then you must use the ad hoc process `LOAD_CURRENT_BUSINESS_DATE_ADHOC` to set it now.

Follow the steps below to perform this transition to nightly batches:

1. Navigate to the **System Options** in POM for the AIF DATA batch schedule.
2. Update the variables **RDE_RunFactVersion** and **RDE_RunFactODIVersion**. By default, they should have a value of `I` as their rightmost input parameter, which is the normal incremental batch run. Change the value to `F`, which will trigger a full snapshot batch run. Do not change any other values already in these options except the letter `I` or `F` at the end.

The RDE_DIM_FLOW_ADHOC process also uses these parameters to extract either full or incremental datasets, if you want to use that process to extract the fact data outside of nightly batch.

3. If you are running the extract outside the Merchandising nightly batch, then disable or skip the interschedule validation jobs (any job starting with RDE_INTERSCHED*) as well as RDE_SETUP_INCRMNTL_CLRPRICE_JOB, RDE_SETUP_INCRMNTL_RESA_JOB, and RDE_SETUP_INCRMNTL_DEALACT_JOB in the RDE portion of the batch flow that will prevent you from running without MFCS nightly jobs. These jobs look for status updates from MFCS nightly jobs and eventually fail if no updates are found.
4. Schedule and run the full Merchandising and AIF nightly batch cycle and validate that all nightly data was processed as expected in your RAP solutions. If you want to validate the RDE extracts prior to running the rest of the nightly batches in RAP (for example, to confirm the full extracts worked as intended) you may place a Hold on one of the first data warehouse load jobs in the schedule, such as BATCH_START_NOTIFICATION_JOB.

There are numerous validations and checkpoints in the AIF DATA batch that aim to prevent improper use of the nightly batch cycle by failing the batch when an issue is detected. These jobs are listed below, along with reasons they might fail if this is your first time running a nightly batch cycle.

Job	Reasons for Failure
RDE_BATCHFINAL_CHKBATCHSTATUSSDE_JOB	Checks the C_LOAD_DATES table for any failed RDE extract jobs. This fails if there is at least one extract job that failed. The purpose is to give you a chance to correct RDE failures before you start loading the RAP data warehouse.
ZIP_FILE_WAIT_JOB	This job is normally disabled when RDE is used; but if it is enabled, it will wait for ZIP files from Object Storage and eventually fail if none are found. If you are not providing any ZIP files, disable the job.
DAT_FILE_VALIDATE_JOB	Associated with the ZIP file jobs, and validates that all required files are present and fails if not. If you are not providing any ZIP files, disable this job.
ETL_REFRESH_JOB	This should be disabled when RDE jobs are in use; it fails if you have it enabled incorrectly. It is a check on C_LOAD_DATES statuses at the start of a non-MFCS batch cycle.
FACT_POSFACT_VALIDATOR_JOB	Checks that the DAY_DT values on all positional fact loads (such as Inventory Position) is equal to the current business date only (positional facts do not support back-posting data) and fails if any issues are found. This usually means that the business date in RAP is not correct (at this point in batch, the RAP business date should equal the MFCS VDATE that was extracted by the RDE jobs in this run).
DIM_CALENDAR_VALIDATOR_JOB	Checks that the incoming calendar is properly formed with no fatal errors in the configuration of fiscal periods. For MFCS data, this can fail if the system options for the MFCS calendar are out of sync with the actual data in the CALENDAR table.
DIM_PROD_VALIDATOR_JOB	Checks that the incoming product hierarchy and item data is properly formed and complete. MFCS data can fail several of the validation rules; review the *AIF Operations Guide* for details.

Job	Reasons for Failure
DIM_ORG_VALIDATOR_JOB	Checks that the incoming location hierarchy and store/ warehouse data is properly formed and complete. MFCS data can fail several of the validation rules; review the *AIF Operations Guide* for details.

Using RDE for Initial Seeding (Gen 1 Architecture)

RDE ad hoc batch programs in the RMFCS 19.x cloud can be used for initial seeding of RAP but the process is different from the 2nd generation architecture, as the integration is through flat files, not direct loads. This section assumes you have already set up your RAP applications, including calendar loads, following the file-based approach documented in Setup and Configuration and Data Loads and Initial Batch Processing. Do not proceed with these steps until you have at least done the initial calendar and partition setup.

For daily batches, integration from RDE (in Merchandising) to RAP occurs automatically as part of the RDE ZIP file upload jobs. RDE will push the ZIP file to the File Transfer Services location used by AI Foundation for incoming data (`ris/incoming` path in FTS). At this point, you have the option to run the AI Foundation batch jobs to use that ZIP or download it from FTS to manually modify it and re-upload it. Once the AI Foundation nightly batches are enabled, you would also enable the batch link connecting AIF to RDE, and then the entire end-to-end process will occur without user intervention.

For initial seeding, you would follow the process below:

1. Set up a full RDE batch (by enabling batch links/dependencies to the RMFCS schedule) and let it run nightly to get the full set of RDE files for dimensions and facts.
2. The file will be pushed automatically to RAP FTS. Download the `RI_RMS_DATA.zip` file from FTS; do not load it into RAP yet.
3. Run the process `SEEDPOSITIONALFACT_ADHOC`, which will extract full snapshots of all positional data, zip them, and push them to the RAP FTS location.
4. Download the `RIHIST_RMS_DATA.zip` file from FTS and copy the full snapshots of positional facts into the `RI_RMS_DATA.zip` file generated by the RDE nightly process (replacing the incremental files that were extracted).
5. Upload the modified RDE nightly ZIP file to RAP FTS at the `ris/incoming` location (same as you would for all nightly batches going forward). Upload any additional ZIP files you need for the nightly batches, such as `ORASE_WEEKLY.zip` or `RAP_DATA.zip`, if you want these other files loaded in the same batch.
6. Advance the ETL business date in AIF to one day before the current batch, if it’s not already set to that date, using the ad hoc process `LOAD_CURRENT_BUSINESS_DATE_ADHOC`. Review any configurations in `C_ODI_PARAM` and `RSE_CONFIG` tables which may have been altered for your historical loads but need updates for nightly batch data. For example, you may want to update `RI_INVAGE_REQ_IND` in `C_ODI_PARAM` if you need calculations of first/ last receipt dates and inventory age from the RMFCS data.
7. Schedule a run of the full AIF nightly batch. Ensure your AIF POM schedule dates for the nightly batch run is aligned with the completed run of RMFCS/RDE, because, from this point forward, the batch schedules will need to remain in sync.

Your transactional facts, such as sales and receipts, should already have history loaded up to this first run of nightly batches, because the next RDE nightly batch will only extract data for the current vdate in RMFCS (for example, it will use the contents of the `IF_TRAN_DATA` daily transaction table for most fact updates besides sales, which come from Sales Audit directly). Once this first AIF batch completes using the full snapshot of positional data, you may prepare for regular nightly batches which will use the incremental extracts from RDE.

The calendar validator job rule `CAL_R2` may cause the batch to fail if this is the first time using Merchandising calendar data directly. This is because the default system calendar in Merchandising does not follow RAP recommendations, which is that the first year of the calendar must be a complete fiscal year. If this happens, verify that the first year of the Merchandising calendar exists much earlier than any actual data in RAP (for example, the Merchandising calendar starts in 2010 but RAP data only exists from year 2020 onwards). If this is confirmed, you may change the validation rule to be a warning instead of an error. Update the table `C_DIM_RULE_LIST` from the Control & Tactical Center. Change the error type for this rule to `W` so that it only throws a warning in the batch program instead of failing, and then restart the failed validator job as needed.

5 Batch Orchestration

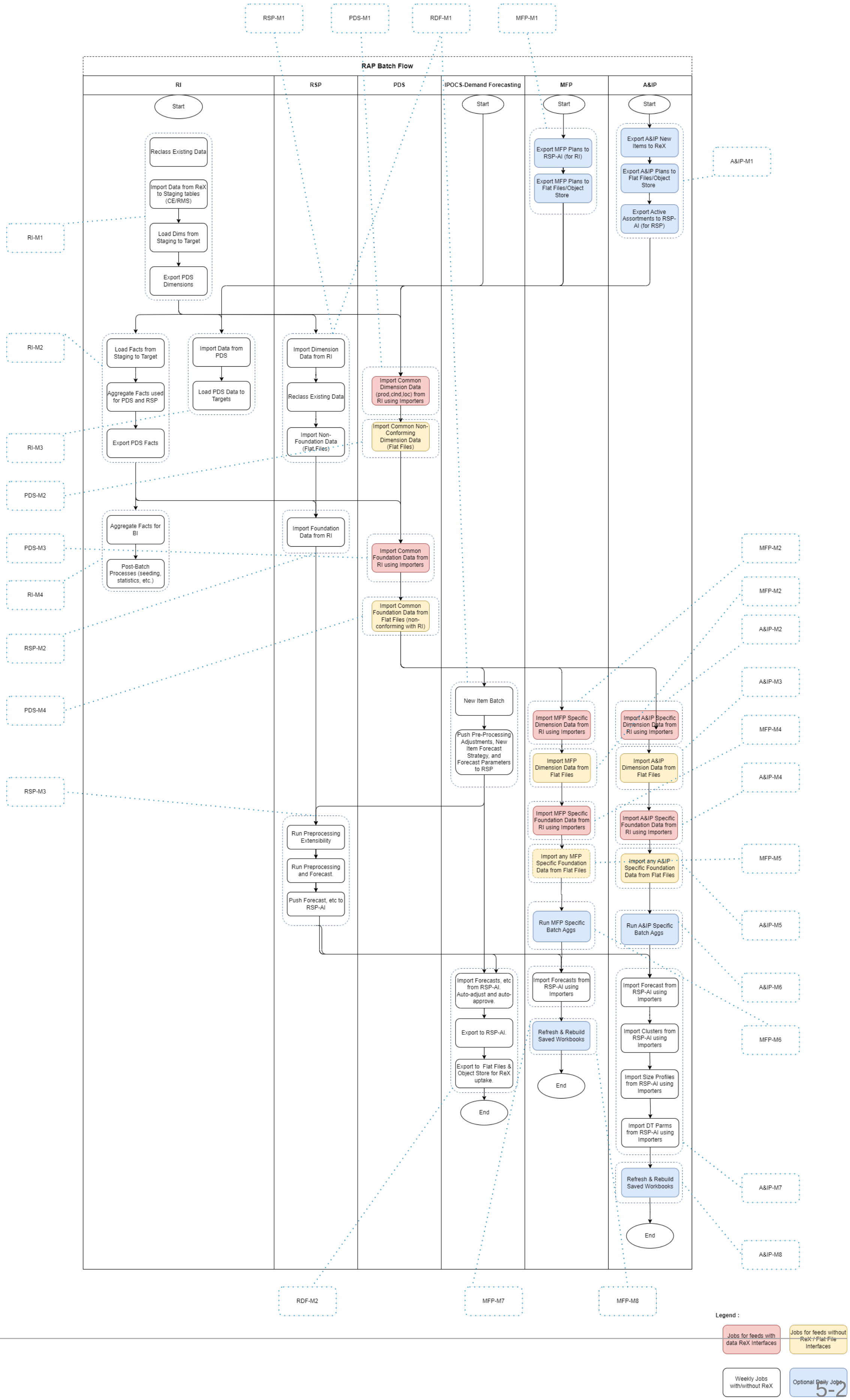
This chapter describes the tools, processes, and implementation considerations for configuring and maintaining the batch schedules used by the Retail Analytics and Planning. This includes nightly, weekly, and ad hoc batch cycles added in the Process Orchestration and Monitoring (POM) tool for each of the RAP applications.

Overview

All applications on the Retail Analytics and Planning have either a nightly or weekly batch schedule. Periodic batches allow the applications to move large amounts of data during off-peak hours. They can perform long-running calculations and analytical processes that cannot be completed while users are in the system, and close out the prior business day in preparation for the next one.

To ensure consistency across the platform, all batch schedules have some level of interdependencies established, where jobs of one application require processes from another schedule to complete successfully before they can begin. The flow diagram below provides a high-level view of schedule dependencies and process flows across RAP modules.

Figure 5-1 Batch Process High-Level Flow



The frequency of batches will vary by application. However, the core data flow through the platform must execute nightly. This includes data extraction from RMS by way of RDE (if used) and data loads into Retail Insights and AI Foundation.

Downstream applications from Retail Insights, such as Merchandise Financial Planning, may only execute the bulk of their jobs on a weekly basis. This does not mean the schedule itself can run weekly (as MFP batch has been run in previous versions); those end-of-week processes now rely on consumption and transformations of data happening in nightly batches. For example, Retail Insights consumes sales and inventory data on a daily basis. However, the exports to Planning (and subsequent imports in those applications) are only run at the end of the week, and are cumulative for all the days of data up to that point in the week.

For this reason, assume that most of the data flow and processing that is happening within the platform will happen every day and plan your file uploads and integrations with non-Oracle systems accordingly.

While much of the batch process has been automated and pre-configured in POM, there are still several activities that need to be performed which are specific to each implementation. TheTable 5-1 table summarizes these activities and the reasons for doing them. Additional details will be provided in subsequent sections of this document

Table 5-1 Common Batch Orchestration Activities

Activity	Description
Initial Batch Setup	By default, most batch processes are enabled for all of the applications. It is the implementer's responsibility to disable batches that will not be used by leveraging the Customer Modules Management screen in Retail Home.
Configure POM Integrations	The POM application supports external integration methods including external dependencies and process callbacks. Customers that leverage non-Oracle schedulers or batch processing tools may want to integrate POM with their existing processes.
Schedule the Batches	Schedules in POM must be given a start time to run automatically. Once started, you have a fixed window of time to provide all the necessary file uploads, after which time the batch will fail due to missing data files.
Batch Flow Details	It is possible to export the batch schedules from POM to review process/job mappings, job dependencies, inter-schedule dependencies, and other details. This can be very useful when deciding which processes to enable in a flow or when debugging fails at specific steps in the process, and how that impacts downstream processing.

Initial Batch Setup

As discussed in Implementation Tools, setting up the initial batches requires access to two tools: POM and Retail Home. Refer to that chapter if you have not yet configured your user accounts to enable batch management activities.

When using POM and Customer Modules Management, it is required to enable or disable the application components based on implementation needs. The sections below describe which modules are needed to leverage the core integrations between applications on the platform. If a module is not present in the below lists, then it is only useful if you are implementing that specific application and require that specific functionality. For example, most modules within

Retail Insights are not required for the platform as a whole and can be disabled if you do not plan to implement RI.

After you make changes to the modules, make sure to synchronize your batch schedule in POM following the steps in Implementation Tools. If you are unsure whether a module should be enabled or not, you can initially leave it enabled and then disable jobs individually from POM as needed.

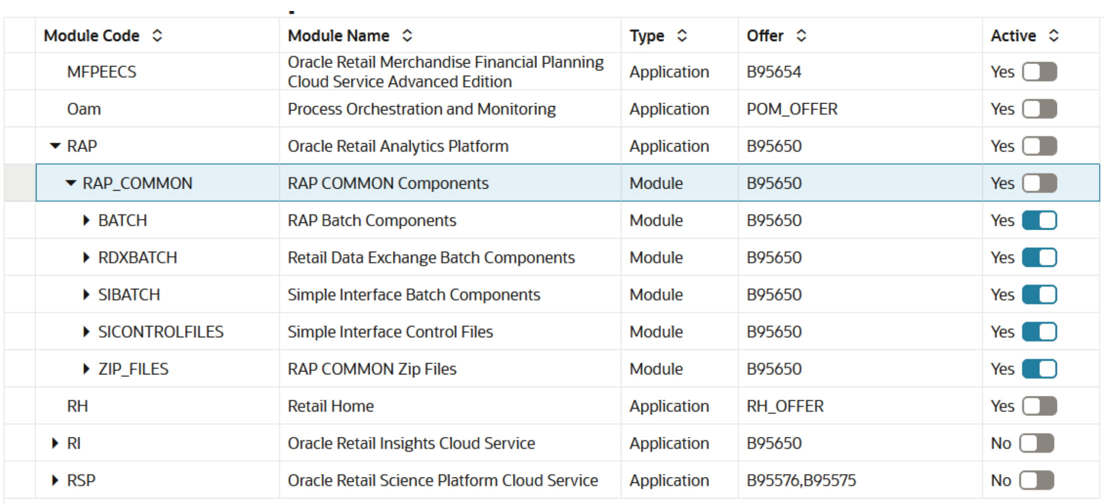
Common Modules

The following table lists the modules that are used across all platform implementations. These modules process core foundation data, generate important internal datasets, and move data downstream for other applications to use. Verify in your environment that these are visible in Customer Modules Management and are enabled.

Table 5-2 RAP Common Batch Modules

Application	Module	Usage Notes
RAP	RAP_COMMON > BATCH	Contains the minimum set of batch processes needed to load data into the platform and prepare it for one or more downstream applications (such as MFP or AI Foundation) when RI is not being implemented. This allows implementers to disable RI modules entirely to minimize the number of active batch jobs.
RAP	RAP_COMMON > RDXBATCH	Contains the batch processes for extracting data from the RAP data warehouse to send to one or more Planning applications.
RAP	RAP_COMMON > ZIP_FILES	Choose which ZIP packages must be present before the nightly batch process begins. The batch only looks for the enabled files when starting and fails if any are not present.
RAP	RAP_COMMON > SIBATCH	Choose which input interfaces will be executed for loading flat files into RAP. Disabled interfaces will not look for or load the associated file even if it is provided.
RAP	RAP_COMMON > SICONTROLFILES	Choose which input files must be present before the nightly batch process begins. Active control files mark a file as required, and the batch fails if it is not provided. Inactive control files are treated as optional and are loaded if their associated job is enabled, but the batch will not fail if they are not provided.
RAP	HISTORY > BATCH	Choose which functional areas require history data or data corrections, either now or after you start batches. Refer to the Module Name column for the matching CSV file names and enable the modules where you plan to provide data. This affects the `ADHOC` processes in the Standalone schedule.
RAP	RAP_COMMON > EGRESS	These modules must be **disabled** for most implementations, unless you are an existing v19 or earlier customer that is working with Oracle to migrate your environments (also known as a Core Update for Planning applications).

Figure 5-2 Example of RAP Common Modules



Module Code	Module Name	Type	Offer	Active
MFPEECS	Oracle Retail Merchandise Financial Planning Cloud Service Advanced Edition	Application	B95654	Yes
Oam	Process Orchestration and Monitoring	Application	POM_OFFER	Yes
RAP	Oracle Retail Analytics Platform	Application	B95650	Yes
RAP_COMMON	RAP COMMON Components	Module	B95650	Yes
BATCH	RAP Batch Components	Module	B95650	Yes
RDXBATCH	Retail Data Exchange Batch Components	Module	B95650	Yes
SIBATCH	Simple Interface Batch Components	Module	B95650	Yes
SICONTROLFILES	Simple Interface Control Files	Module	B95650	Yes
ZIP_FILES	RAP COMMON Zip Files	Module	B95650	Yes
RH	Retail Home	Application	RH_OFFER	Yes
RI	Oracle Retail Insights Cloud Service	Application	B95650	No
RSP	Oracle Retail Science Platform Cloud Service	Application	B95576,B95575	No

If you are implementing some or all of AI Foundation or Retail Insights, then there are some additional modules to review. These modules may or may not be required, as they are based on which interface files you plan to load as part of the nightly batch process.

Application	Module	Usage Notes
RAP	AIF>CONTROLFILES	Enable only the files you plan to send based on your AI Foundation/Retail Insights implementation plan. Refer to the Interfaces Guide for details. This is only used for non-foundation or legacy DAT files. This directly affects what files are required to be in the zip files, when an AIF DATA Daily Batch is executed. Required files that are missing will fail the batch.
RAP	AIF > BATCH	Enable `RSP_REQUIRED` if you are implementing any AI Foundation module, plus others based on your application needs. For example, if you are loading sales then enable `RSP_SALES`. If you are loading inventory, then enable `RSP_INVPOS`, and so on.

After setting up the common modules and syncing with POM, ensure that certain critical batch processes in the AIF DATA schedule (which is used by all of RAP) are enabled in Batch Monitoring. This can be used as a check to validate the POM sync occurred:

- RESET_ETL_THREAD_VAL_STG_JOB
- TRUNCATE_STAGE_TABLES_JOB (unless you are using RDE with RMFCS v22+, then this must be disabled instead)
- DELETE_STATS_JOB
- RI_UPDATE_TENANT_JOB

Some of these jobs begin in a disabled state in POM (depending on the product version) so the POM sync should ensure they are enabled. If they are not enabled after the POM sync, be sure to enable them before attempting any batch runs.

Additionally, there are certain jobs that must remain disabled unless advised to enable them by Oracle. Make sure the following jobs are disabled in the AIF DATA schedule after syncing with POM:

- OBIEE_CACHE_CLEAR_JOB
- ODI_LOG_EXTRACTOR_JOB
- ODI_LOG_LOADER_JOB
- If you are not providing a file named `RA_SRC_CURR_PARAM_G.dat` in all ZIP uploads, disable `BATCH_VALIDATION_JOB`, `RA_SRC_CURR_PARAM_G_COPY_JOB`, and `RA_SRC_CURR_PARAM_G_STG_JOB`
- If you are using stock ledger in RI, only one of the following can be used and the other must be disabled: `W_RTL_STCKLDGR_SC_LC_MH_F_GREG_JOB`, `W_RTL_STCKLDGR_SC_LC_MH_F_JOB`

RI Modules

The following table lists modules within the Retail Insights offer codes which may be used if you are implementing any part of Retail Insights. Disable the RI module entirely if you are not implementing Retail Insights at this time, because all your batch configurations should be covered by the RAP common modules.

Table 5-3 RI Batch Modules

Application	Module	Usage Notes
RI	RMI > BATCH	Enable some or all of the modules in this section to use Merchandising data in Retail Insights.
RI	RCI > BATCH	Enable some or all of the modules in this section to use Customer and Consumer data in Retail Insights.

AI Foundation Modules

The following table lists modules within the AI Foundation applications which may be used by one or more other RAP applications, in addition to the common modules from the prior section. This primarily covers forecasting and integration needs for Planning application usage. It is important to note that the underlying process for generating forecasts leverages jobs from the Lifecycle Pricing Optimization (LPO) application, so you will see references to that product throughout the POM batch flows and in Retail Home modules.

Table 5-4 AI Foundation Shared Batch Modules

Application	Module	Usage Notes
AI Foundation	FCST > Batch	Collection of batch jobs associated with foundation loads, demand estimation, and forecast generation/export. The modules under this structure are needed to get forecasts for Planning.

To initialize data for the forecasting program, use the ad hoc POM process `RSE_MASTER_ADHOC_JOB` described in Sending Data to AI Foundation. After the platform is initialized, you may use the Forecast Configuration user interface to set up and run your initial forecasts. For complete details on the requirements and implementation process for forecasting, refer to the *Retail AI Foundation Cloud Services Implementation Guide*.

For reference, the set of batch programs that should be enabled for forecasting are listed below (not including foundation data loads common to all of AI Foundation). Enable these by syncing POM with MDF modules, though it is best to validate that the expected programs are enabled after the sync.

> **Note:**
>
> Jobs with PMO in the name are also used for Lifecycle Pricing Optimization and are shared with the forecasting module.

Job Name
PMO_ACTIVITY_LOAD_START_JOB
PMO_ACTIVITY_STG_JOB
PMO_ACTIVITY_LOAD_JOB
PMO_ACTIVITY_LOAD_END_JOB
PMO_CREATE_BATCH_RUN_JOB
PMO_RUN_EXEC_SETUP_JOB
PMO_RUN_EXEC_START_JOB
PMO_RUN_EXEC_PROCESS_JOB
PMO_RUN_EXEC_END_JOB
RSE_CREATE_FCST_BATCH_RUN_JOB
RSE_FCST_BATCH_PROCESS_JOB
RSE_FCST_BATCH_RUN_END_JOB
RSE_CREATE_FCST_BATCH_RUN_ADHOC_JOB
RSE_FCST_BATCH_PROCESS_ADHOC_JOB

There are also two processes involved in forecast exports to MFP, one as part of weekly batch and the other as an ad hoc job which you can run during implementation.

Job Name
RSE_MFP_FCST_EXPORT_JOB

Job Name
RSE_MFP_FCST_EXPORT_ADHOC_JOB

Maintenance Cycles

The AIF DATA schedule also includes a standalone maintenance batch flow called `RI_MAINTENANCE_ADHOC`. You need to enable and schedule this process to run nightly, sometime prior to the main nightly batch.

1. From Batch Administration, go into the AIF DATA schedule Standalone tab and locate the `RI_MAINTENANCE_ADHOC` flow. If the jobs are disabled, then enable them now.
2. Go into Schedule Administration and add a new schedule for the flow, enable it, and schedule it to run once every day. It is recommended to start it several hours prior to the actual nightly batch; for example, it can run starting at 8 pm local time if your nightly batch starts at 2 am the following morning. These processes should not impact normal user activity in the applications and are safe to run even if users are still logged in.
3. From Batch Monitoring, restart the AIF DATA schedule to apply the changes.

The maintenance cycle is responsible for creating table partitions, repairing unusable indexes, purging old exports to PDS from the RDX integration tables, purging old log files, and purging certain records relating to deleted items and locations. All of these activities are necessary to ensure the data warehouse operates efficiently over time as data continues to accumulate in the system. These jobs are kept outside of the nightly batch flow because there is a chance they will need several hours to run in some instances, such as when a large number of new partitions need to be created at the start of a fiscal quarter or a particularly large number of log files have to be purged.

Batch Setup Example

This section will guide you through an example of setting modules in Retail Home and syncing with POM. A basic implementation of MFP with forecasting in AIF might send all the following foundation data files (and their CTX files):

- CALENDAR.csv
- PRODUCT.csv
- ORGANIZATION.csv
- EXCH_RATE.csv
- SALES.csv
- INVENTORY.csv
- RECEIPT.csv
- TRANSFER.csv
- MARKDOWN.csv
- ADJUSTMENT.csv
- ORDER_HEAD.csv
- ORDER_DETAIL.csv
- RTV.csv
- RA_SRC_CURR_PARAM_G.dat

These files will be bundled into a single ZIP file named `RAP_DATA.zip`. To configure the Customer Modules for this batch implementation, perform the following steps:

1. At the top level, you may disable the RI module entirely (if it's visible), because you are not implementing that solution
2. Enable and expand the RAP module. Within the `RAP_COMMON` sub-module, enable only the following components and disable the rest:
   - RAP>RAP_COMMON>ZIP_FILES: RAP_DATA_ZIP
   - RAP>RAP_COMMON>SICONTROLFILES: RAP_SI_DIM_CALENDAR, RAP_SI_DIM_EXCHANGE_RATES, RAP_SI_DIM_ONORDER, RAP_SI_DIM_ORGANIZATION, RAP_SI_DIM_PRODUCT, RAP_SI_FACT_ADJUSTMENT, RAP_SI_FACT_INVENTORY, RAP_SI_FACT_MARKDOWN, RAP_SI_FACT_ORDER_DETAIL, RAP_SI_FACT_RECEIPT, RAP_SI_FACT_RTV, RAP_SI_FACT_SALES, RAP_SI_FACT_TRANSFER
   - RAP>RAP_COMMON>SIBATCH: RAP_SI_INVADJ, RAP_SI_INVPOS, RAP_SI_INVRECEIPT, RAP_SI_INVRTV, RAP_SI_INVTRANSFER, RAP_SI_MARKDOWN, RAP_SI_ONORDER, RAP_SI_PO, RAP_SI_REQUIRED, RAP_SI_SALES
   - RAP>RAP_COMMON>RDXBATCH: RAP_RDX_INVADJ, RAP_RDX_INVPOS, RAP_RDX_INVECEIPT, RAP_RDX_INVRTV, RAP_RDX_INVTRANSFER, RAP_RDX_MARKDOWN, RAP_RDX_PO, RAP_RDX_REQUIRED, RAP_RDX_SALES
   - RAP>RAP_COMMON>BATCH: RAP_INVADJ, RAP_INVPOS, RAP_INVRECEIPT, RAP_INVRTV, RAP_INVTRANSFER, RAP_MARKDOWN, RAP_PO, RAP_REQUIRED, RAP_SALES
3. To use AIF for forecasting, you will also need certain parts of the AI Foundation modules (named RSP in Retail Home). Enable the **RSP** root module, expand it, and enable the **FCST** module. All options under **FCST** should also be enabled. Disable all other RSP modules here.
4. If you are integrating with Merchandising using the RDE batch jobs, then you are likely not providing flat files for most things, and the setup process will be different. Refer to Integration with Merchandising to understand the batch setup process in more detail.

Once all changes are made, make sure to **Save** the updates using the button below the table. Only after saving the changes will they be available to sync with POM.

The next set of steps will be performed from POM:

1. Go to the Batch Administration screen for each schedule (**Nightly** tabs of AIF DATA and AIF APPS schedules) and disable ALL jobs in the Nightly schedule. By disabling all jobs, the next step will selectively enable just what is needed.
2. Click **Sync with MDF** in each schedule. Wait for the sync to complete (it will take some time and the screen will prevent any updates while syncing).
3. Review the list of jobs that are now enabled to verify that it is correct (for example, that the jobs relating to **PRODUCT** file loads are all enabled similarly to when you ran historical data loads).
4. Inter-schedule dependencies must also be enabled at this time from Batch Administration screen. You will need to locate the jobs having dependencies and turn them on within the job details screen. Update the following jobs in each schedule to enable any inter-schedule dependencies they have:

Schedule	Job Name
AIF APPS	RSE_WEEKLY_INPUT_FILES_START_JOB
RPASCE	MFP_POST_DATA_IMP_START_JOB
RPASCE	RPASCE_DATA_IMP_START_JOB
RPASCE	RPASCE_HIER_IMP_START_JOB

5. Go to the Batch Monitoring screen. If you have a schedule already open for a past business date, click **Close Schedule**. If you are already on the current business date then just click **Restart Schedule** and skip the next step.
6. Change the POM business date to the date you will be loading nightly batch data for (for example if your data files will have data for 2/25/2023 then that should also be the business date). Open a new schedule using the provided button.
7. Enable and schedule the batches to run from Schedule Administration. The inter-schedule dependencies will ensure downstream jobs are not run until the necessary dependencies are complete (for example, RSP jobs will wait for RI loads to complete).

   > **Note:**
   >
   > Updating the schedule times also requires a Restart Schedule action to be performed afterwards.

8. If you have not already enabled the schedule for the `RI_MAINTENANCE_ADHOC` flow, make sure to do that now.
9. Make sure the business date in the data warehouse is one day prior to the first day of data you plan to load through the nightly batch. If necessary, use the standalone POM process `LOAD_CURRENT_BUSINESS_DATE_ADHOC` to advance the business date. For example, if your first nightly batch is loading data for `2023-04-30` then you must ensure the data warehouse is currently on business date `2023-04-29`. The nightly batch will automatically advance the date starting with `2023-04-30`.
10. If you chose not to provide the `RA_SRC_CURR_PARAM_G.dat` file for ensuring proper business date validation, then go back to Batch Administration in the AIF DATA nightly schedule and disable `BATCH_VALIDATION_JOB`, `RA_SRC_CURR_PARAM_G_COPY_JOB`, and `RA_SRC_CURR_PARAM_G_STG_JOB`.

The next set of steps may or may not be needed depending on your business calendar configuration:

1. From the Batch Administration Nightly schedules, locate the column for **Days of the Week**. By default, weekly jobs may run on either Saturday or Sunday (varies by process). You must align the weekly jobs to your week-ending dates for anything involved in integration or calculations, such as RDX and PDS batch jobs.
2. For example, if you wish to set your week-ending date as Saturday for all jobs involved in **RI > Planning integrations**, filter the AIF DATA Schedule job names using each of the following codes: PDS, RDX
3. For each set of PDS or RDX jobs, look at the Days of the Week value. Anything that is not set to run Saturday should be modified by editing the job record and changing the day to Saturday. You might also want some jobs to run daily, in which case you select all days of the week here.
4. Repeat this process in the RPASCE schedule, moving any jobs to the desired **Day of the Week** value.

5. When all changes are done, be sure to **Restart Schedule** from the Batch Monitoring screen.

Lastly, ensure that the following jobs are disabled in the AIF DATA schedule after all other setup is done, as they should not be used in the current version:

- OBIEE_CACHE_CLEAR_JOB
- ODI_LOG_EXTRACTOR_JOB
- ODI_LOG_LOADER_JOB

You are now ready to begin running your nightly batches. As soon as you run the first batch cycle, you must continue to run AIF DATA and AIF APPS batch cycles every night in sequence. You must not skip any days, because some jobs have internal processing based on the day of week that they execute and skipping days will prevent proper operation of these jobs. If you are not providing daily fact data, then you must still run the daily batches with a full set of dimension files, such as products and locations; you can never run a batch without dimension files present and populated with data. The recommended approach is for the source system providing data files to always package and upload a nightly ZIP file every day even when no data is changing. The uploaded ZIP should contain the dimension data plus empty fact files where daily data is not being generated.

Adjustments in POM

While the bulk of your batch setup should be done in Retail Home, it may be necessary to fine-tune your schedule in POM after the initial configuration is complete. You may need to disable specific jobs in the nightly schedules (usually at Oracle's recommendation) or reconfigure the ad hoc processes to use different programs. The general steps to perform this activity are:

1. From Retail Home, click the link to navigate to POM or go to the POM URL directly if known. Log in as a batch administrator user.
2. Navigate to the Batch Administration screen.
3. Select the desired application tile, and then select the schedule type from the nightly, recurring, or standalone options.
4. Search for specific job names, and then use the **Enabled** option to turn the program on or off.
5. From the Batch Monitoring screen, click **Restart Schedule** to apply the changes.

> **Note:**
>
> If you sync with MDF again in the future, it may re-enable jobs that you turned off inside a module that is turned on in Retail Home. For that reason, the module configuration is typically used only during implementation, then POM is used once you are live in production.

Managing Multiple Data Sources

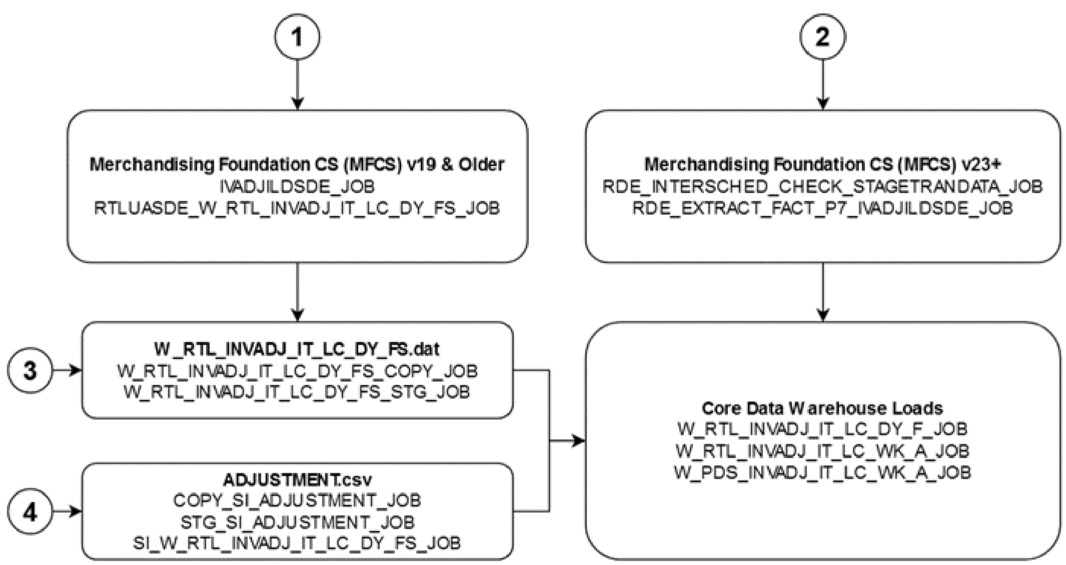
Some implementations require sourcing fact data from more than one source. For example, your inventory data may be coming from Merchandising Foundation CS (MFCS) but your sales data will come from an external 3rd-party application. In such cases, it will be necessary to manually configure your batch schedule for AIF DATA to selectively enable the jobs for your chosen data flow and disable all other paths for that data. The diagrams in this section highlight each possible data flow from an outside source into the AIF DATA data warehouse.

Each numbered entry point into the data flow is mutually exclusive, you cannot run multiple data flows at the same time or it can result in duplicated or invalid data being loaded. For example, if you are entering the diagram from point 1, you would want to ensure all AIF DATA jobs along that path are enabled, and also disable the jobs for unused entry points (2 and 4). The entry points in every diagram are explained in more detail below.

Entry Point	Explanation
1	This is the data flow for Retail Data Extractor for MFCS v19 or earlier. In this flow, RDE is a separate module that runs on the MFCS server and POM instance and creates flat files to export to AIF DATA. This data flow includes on-premise instances of MFCS/RDE.
2	This is the data flow for Retail Data Extractor for MFCS v23 or later. In this flow, RDE jobs are a part of the AIF DATA batch schedule and will directly read from MFCS and insert to AIF DATA staging tables without using any files.
3	This is the data flow for legacy AIF DATA flat file formats, which use the same file structures as RDE v19 or earlier. New implementations will generally not use this entry point for foundation fact data.
4	This is the data flow for simplified foundation interfaces in RAP v23 or later that use CSV file formats. When data is not coming from MFCS, it should usually enter here.

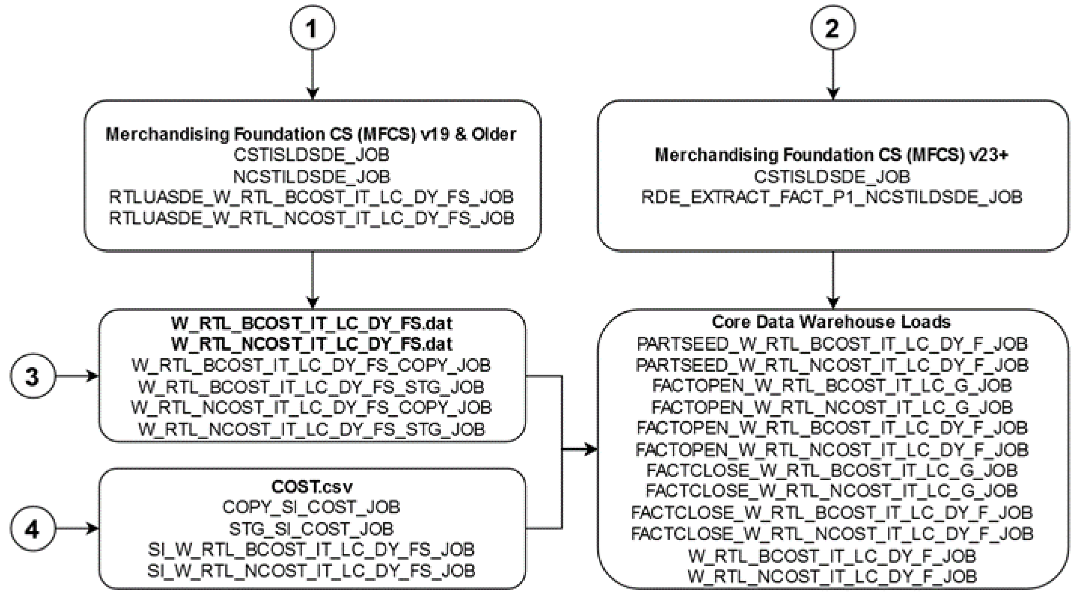
Adjustments

This is the data flow diagram for inventory adjustment transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



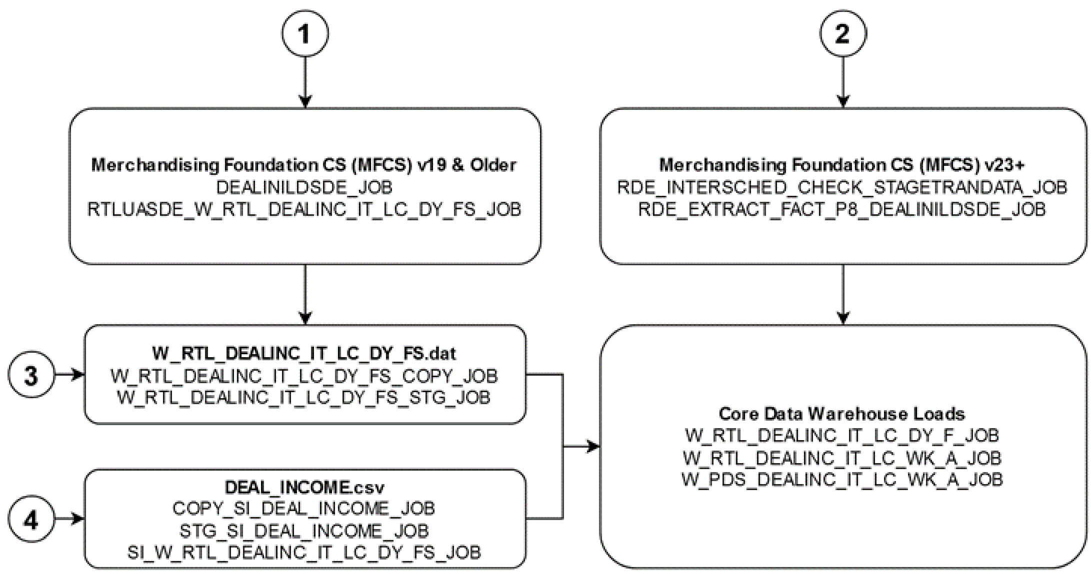
Costs

This is the data flow diagram for base cost and net costs. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



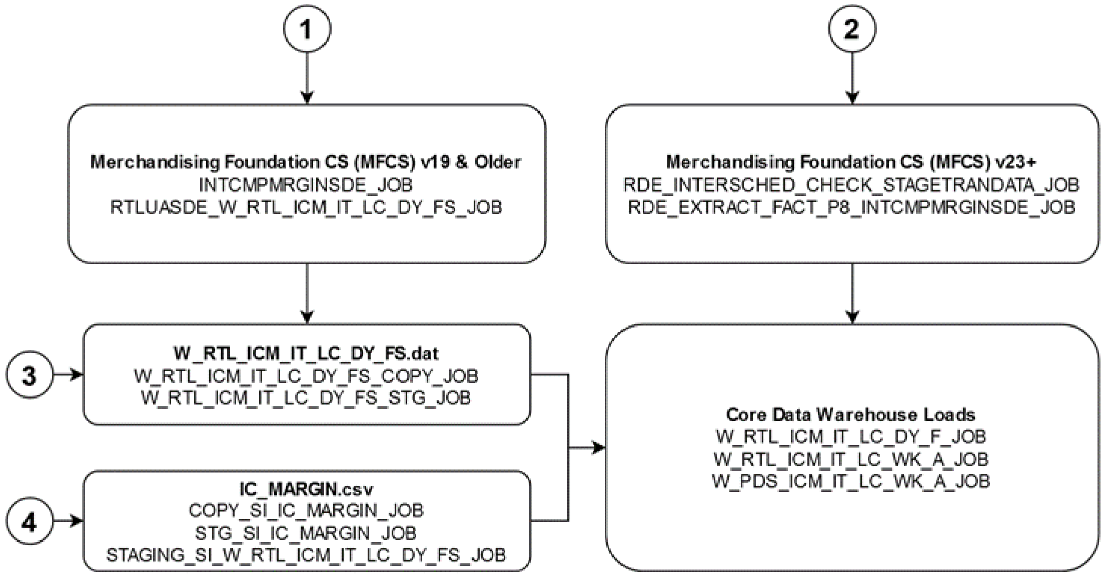
Deal Income

This is the data flow diagram for deal income transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



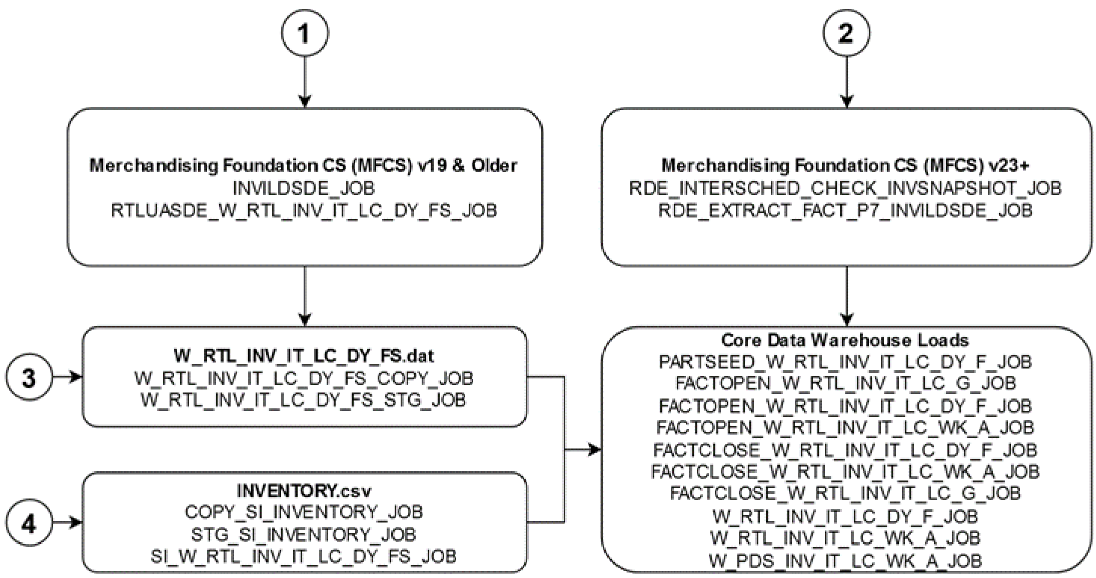
Intercompany Margin

This is the data flow diagram for intercompany margin transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



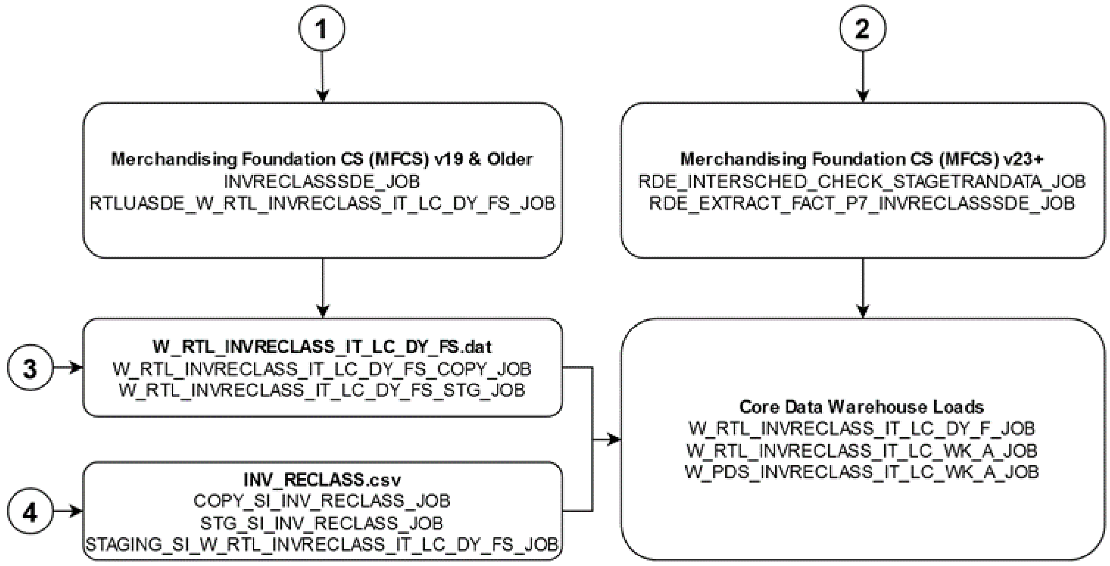
Inventory Position

This is the data flow diagram for inventory positions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



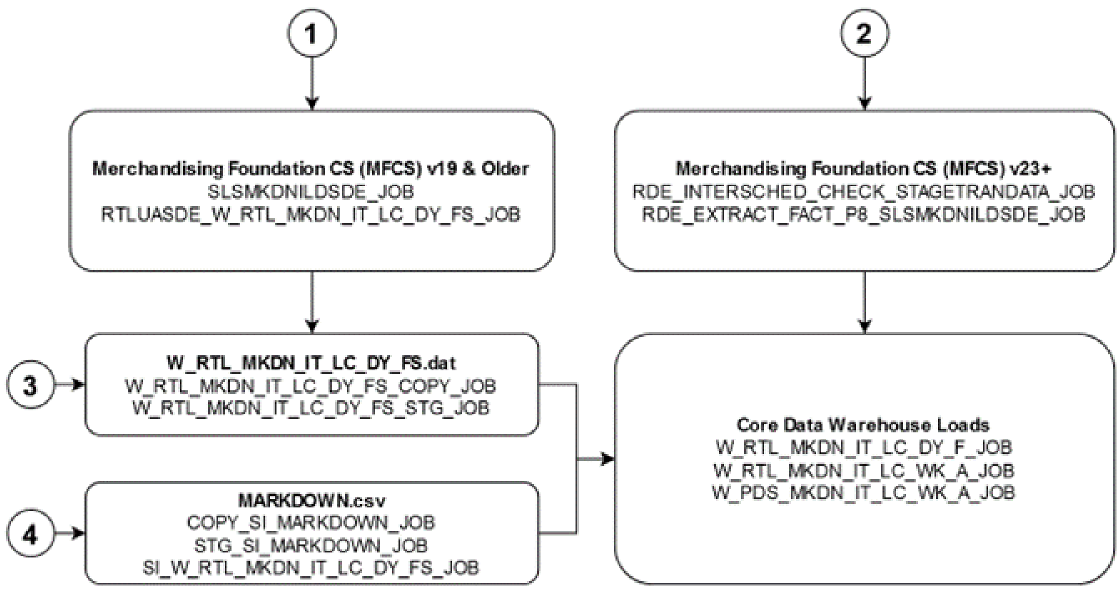
Inventory Reclass

This is the data flow diagram for inventory reclass transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



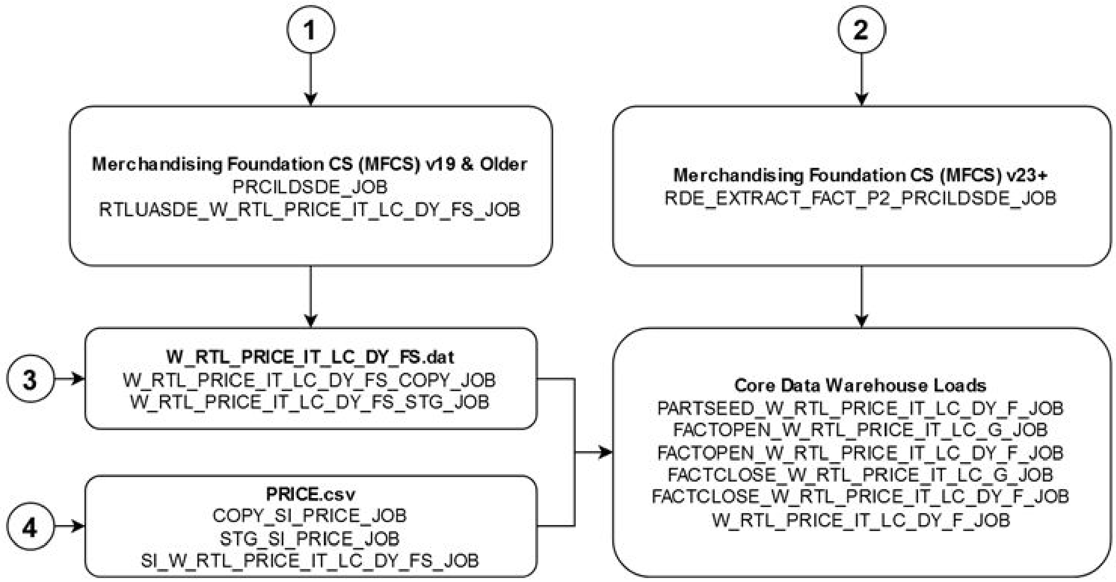
Markdowns

This is the data flow diagram for markdown transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



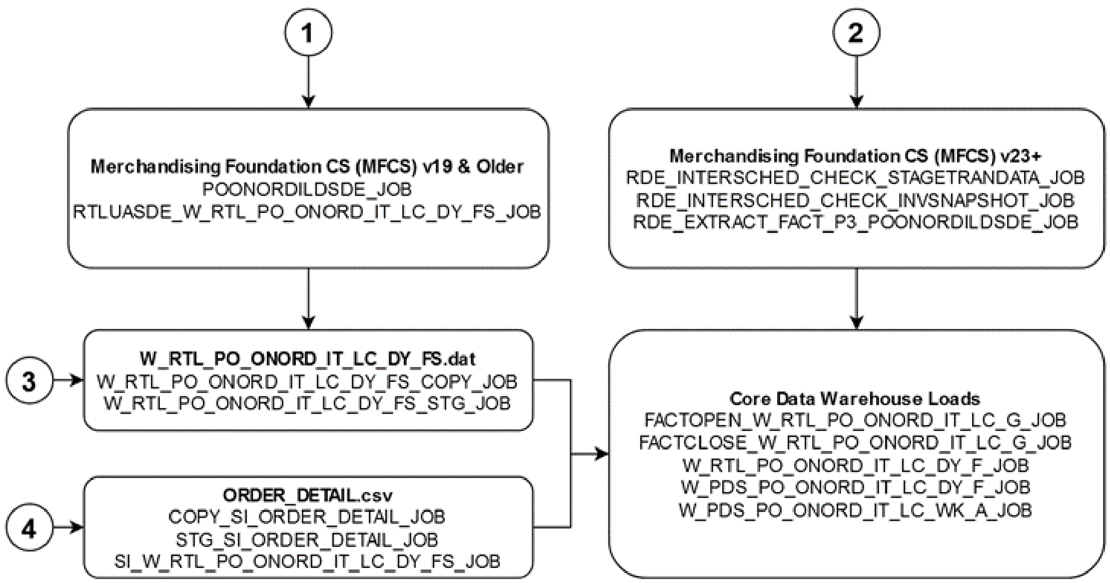
Prices

This is the data flow diagram for daily price updates. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



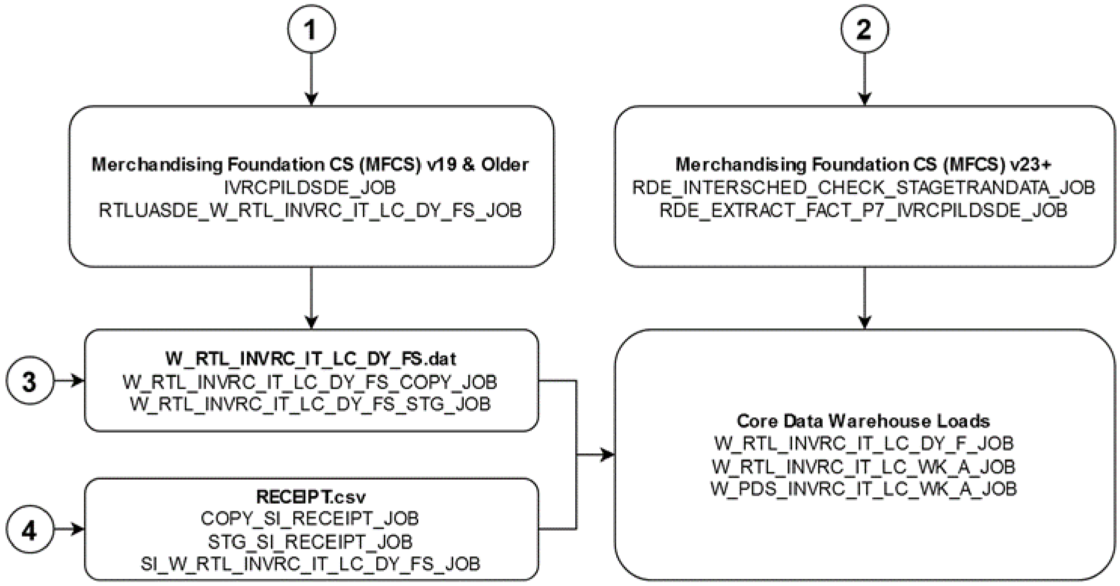
Purchase Orders

This is the data flow diagram for purchase order updates. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



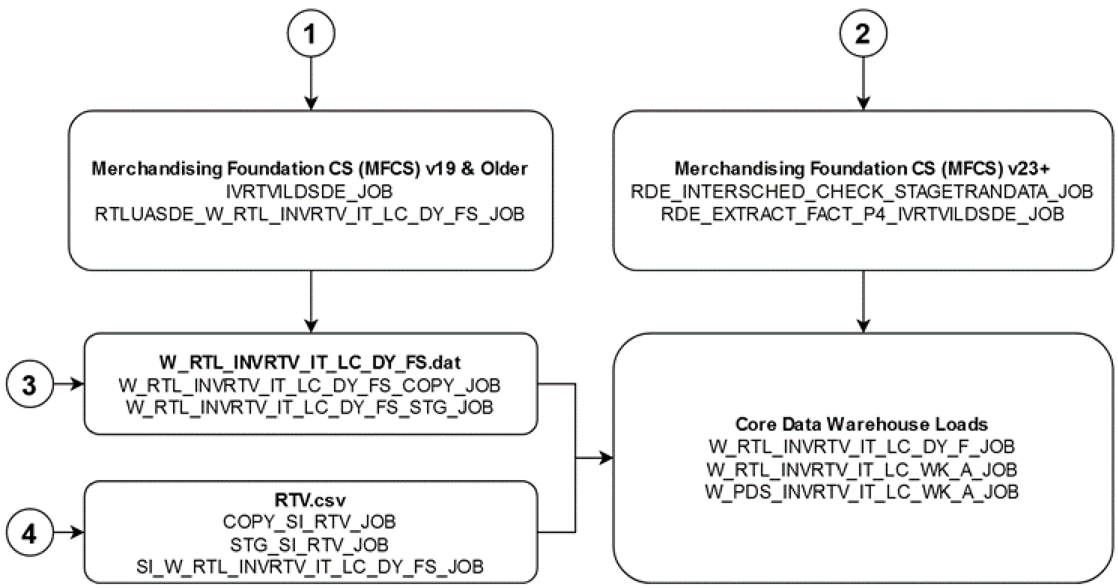
Receipts

This is the data flow diagram for inventory receipt transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



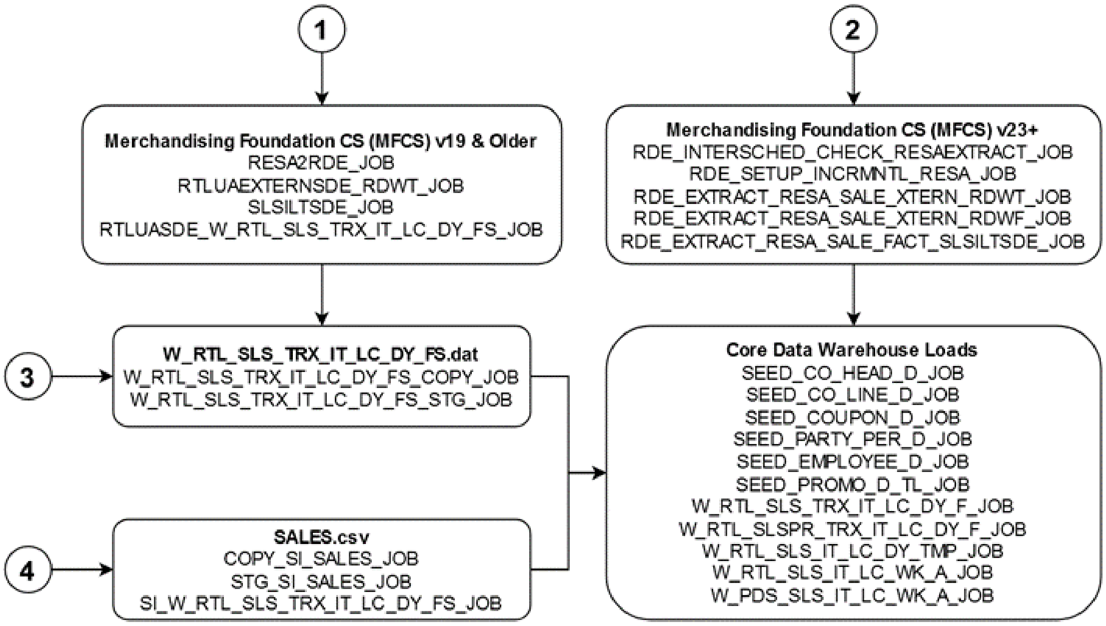
Returns to Vendor

This is the data flow diagram for inventory returns to vendor. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



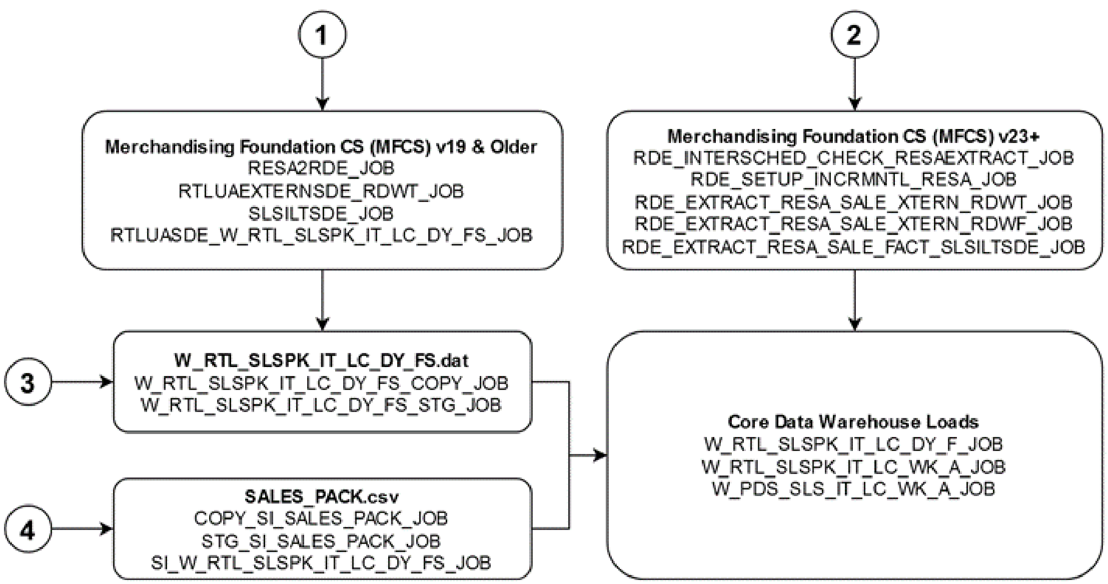
Sales

This is the data flow diagram for sales transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



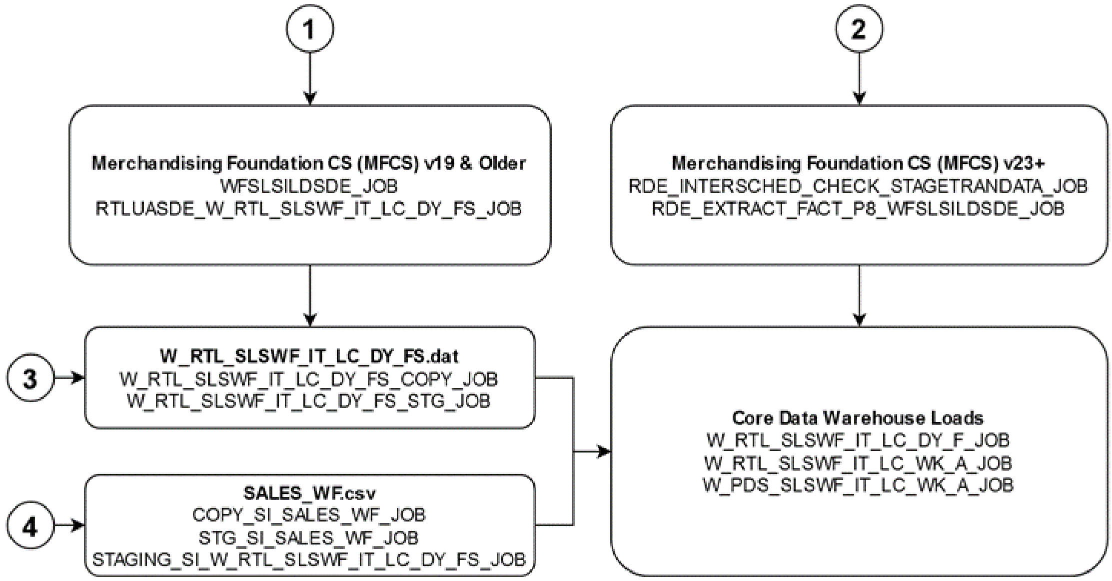
Sales Pack

This is the data flow diagram for sales transactions at component level that are spread down from pack item sales. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



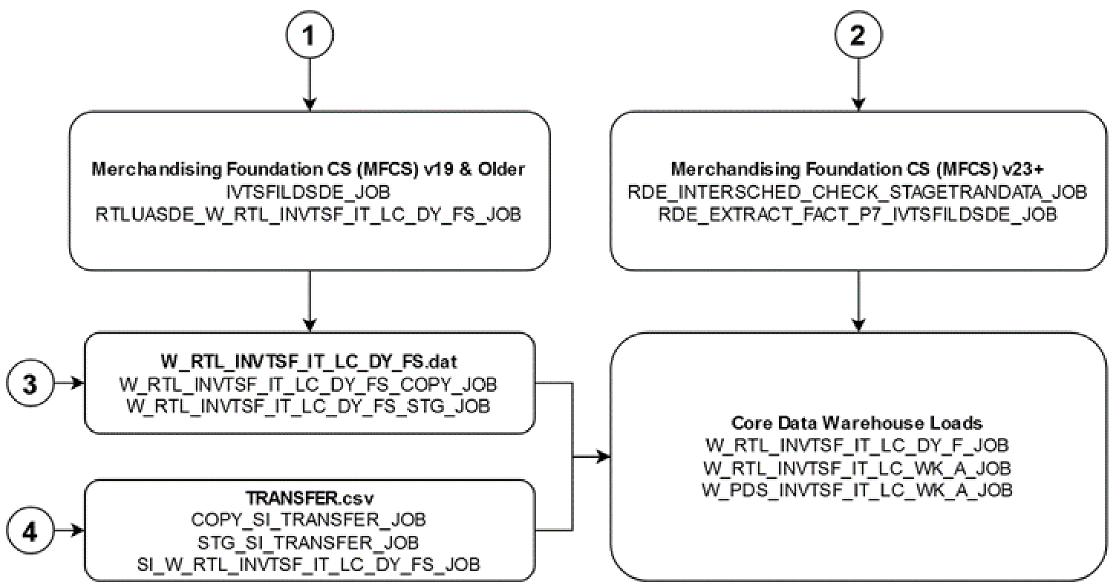
Sales Wholesale

This is the data flow diagram for wholesale and franchise sales transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



Transfers

This is the data flow diagram for inventory transfer transactions. Enable only the jobs for your chosen load method and disable the jobs for all other entry points.



Configure POM Integrations

Retailers often have external applications that manage batch execution and automation of periodic processes. They might also require other downstream applications to be triggered automatically based on the status of programs in the Retail Analytics and Planning. POM supports the following integrations that should be considered as part of your implementation plan.

Table 5-5 POM Integrations

Activity	References
Trigger RAP batches from an external program	*POM Implementation Guide* > Integration > Invoking Cycles in POM
Trigger external processes based on RAP batch statuses	*POM Implementation Guide* > Integration > External Status Update
Add external dependencies into the RAP batch to pause execution at specific points	*POM Implementation Guide* > Integration > External Dependency

Schedule the Batches

Once you are ready to begin regularly scheduled batch processing, you must log in to POM to enable each batch cycle and provide a start time for the earliest one in the batch sequence. The general steps are:

1. Log into POM as an administrator user.
2. Access the Scheduler Administration screen.
3. Select each available tile representing a batch schedule and select the **Nightly** batch option. AIF DATA and AIF APPS schedules must run every night once they are enabled; you cannot run them weekly or skip any days.
4. Edit the rows in the table to enable each batch and enter a start time. You can start all of the batches with similar times or stagger them apart based on how long you think they need to wait before their dependencies will be fulfilled by the linked applications. After any change to the schedule times, you must also **Restart the Schedule** in Batch Monitoring for it to take effect.
5. For each batch after AIF DATA that must execute, you must verify the inter-schedule dependencies are enabled for those batches. You can find the dependencies and enable them by going to the Batch Monitoring screen and selecting each batch's Nightly set of jobs. Click the number in front of Inter-Schedule Dependencies and click **Enable** to the right of each displayed row if the current status is Disabled. This should change the status of the row to Pending.

Refer to the *POM User Guide* for additional information about the Scheduler screens and functionality. Scheduling the batch to run automatically is not required if you are using an external process or scheduler to trigger the batch instead.

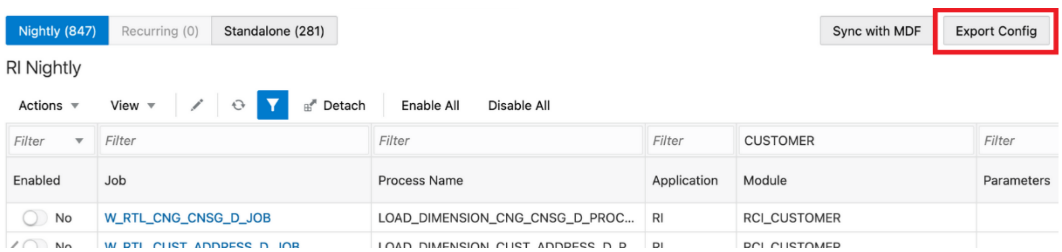
For the AIF DATA nightly batch (which consumes the foundation input files for platform data), the batch will first look for all the required input ZIP packages configured through Customer Modules Management. The batch will wait for 4 hours for all required files, after which it will fail with an error in POM. When choosing a start time for the batch, it is best to start the 4-hour window an hour before you expect to have all the files uploaded for it to use. The batch performs reclassification of aggregate tables as its first step, which can take the full hour to complete in the case of very large reclassifications. This still provides 3+ hours during which you can schedule all file uploads and non-Oracle integration processes to occur.

The other application batches, such as those for the AI Foundation modules, begin automatically once the required Retail Insights batch jobs are complete (assuming you have enabled those batches). Batch dependencies have been pre-configured to ensure each downstream process begins as soon as all required data is present.

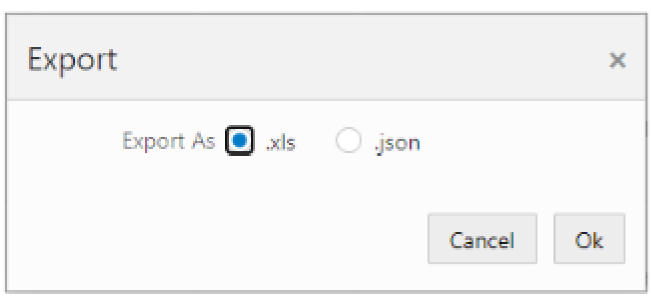
Batch Flow Details

POM allows you to export the full batch schedule configuration to a spreadsheet for review. This is an easier way to see all of the jobs, processes, and dependencies that exist across all batch schedules that you are working with on the platform. Perform the following steps to access batch configuration details:

1. From Retail Home, click the link to navigate to POM, or go to the POM URL directly if known. Log in as a batch administrator user.
2. Navigate to the Batch Administration screen.
3. Select the desired application tile, and then select the schedule type from the nightly, recurring, or standalone options.
4. Click the **Export Config** button to download the schedule data.



5. Choose your preferred format (XLS or JSON).
   - XLS is meant for reviewing it visually.
   - JSON is used for importing the schedule to another environment.
6. Save the file to disk and open it to review the contents.



For more details about the tabs in the resulting XLS file, refer to the *POM User Guide*, “Export/ Import Schedule Configuration”.

Planning Applications Job Details

Planning applications such as Merchandise Financial Planning schedule jobs through POM, and run ad hoc tasks directly using Online Administration Tasks (OAT) within the application. POM is the preferred approach for scheduling jobs in RAP for managing interdependencies with jobs from RI and AI Foundation. Refer to the application Administration Guides for more details on scheduling tasks through OAT.

Planning applications also use a special Batch Framework that controls jobs using batch control files. You may configure these batch control files for various supported changes within the application. The POM schedule for Planning internally calls an OAT task controlled by the batch control entries. A standard set of nightly and weekly jobs for Planning are defined to schedule them in POM. You also have the option to disable or enable the jobs either directly through POM, or by controlling the entries in batch control files. Refer to the Planning application-specific Implementation Guides for details about the list of jobs and how jobs can be controlled by making changes to batch control files.

Reprocessing Nightly Batch Files

When a nightly batch fails due to a corrupt or improperly formed data file, you may want to re-send just the fixed data file and not the entire nightly ZIP upload. An ad hoc POM process is available for uploading individual data files and appending them to the current nightly batch run. The process is called `REPROCESS_ZIP_FILE_PROCESS_ADHOC`. It expects a file `RI_REPROCESS_DATA.zip` to be uploaded using File Transfer Services (FTS). The ad hoc program will import and unpack the ZIP file and directly move any contents into the staging area with all existing data files sent from a previous upload. No existing file will be deleted; it will overwrite only the files provided in `RI_REPROCESS_DATA.zip`.

Once a new file has been placed, you will still need to re-run the jobs to import that file. Or, if a `COPY` job for that file is what failed, re-run that job and the batch will resume from there.

6

Data Processing and Transformations

The Retail Analytics and Planning is a data-driven set of applications performing many transformations and calculations as part of normal operations. Review this chapter to learn about the most common types of data transformation activities occurring within the platform that may impact the data you send into the platform and the results you see as an end user.

Data Warehouse Aggregate Tables

The RAP data warehouse (used to maintain all foundation data for the RAP applications) accepts most data at a common base intersection of item, location, and date. However, downstream applications such as Planning or AI Foundation may need this data at higher levels of aggregation than that. To support those data needs, the data warehouse pre-aggregates the incoming fact data to certain higher levels, depending on the requirements of the consuming application. For Planning purposes, all fact data is aggregated to a common level of item, location, and fiscal week before it is exported for downstream consumption. Review the sections below to learn more about how data moves through the foundation data warehouse for all our applications to use.

Table Structures

If you are currently loading data into the data warehouse using AIF DATA batch jobs and need to access database tables for debugging or validation purposes, there are naming and format conventions used on each aggregate table. A base intersection table is abbreviated using the following notations:

Table 6-1 Data Warehouse Base Fact Structure

Table Name Component	Explanation
W_	Most data warehouse tables start with a “W” to denote a core data warehouse object.
IT	Abbreviation for Item
LC	Abbreviation for Location
DY	Abbreviation for Day
WK	Abbreviation for Week
_D	Dimensional tables end with “D”
_F	Base intersection fact tables end with “F”
_A	Aggregate tables end with “A”

Using the above notation, you may interpret the table `W_RTL_SLS_IT_LC_WK_A` as “Sales aggregate table at the item/location/week intersection”.

Key Columns

Most fact tables in the data warehouse use the same key column structure, which consists of two types of internal identifiers. The first identifier is referred to as a WID value. The WID on a fact table is a foreign key reference to a dimension table's `ROW_WID` column. For example, a `PROD_WID` column in a sales table is referring to the `ROW_WID` on `W_PRODUCT_D` (the product dimension table). Joining the WIDs on a fact and a dimension will allow you to look up user-facing descriptors for the dimensions, such as the product number.

The second identifier is known as `SCD1_WID` and refers to slowly changing dimensions, which is a common data warehousing concept. The IDs on the `SCD1_WID` columns are carried forward through reclassifications and other dimensional changes, allowing you to locate a single product throughout history, even if it has numerous records in the parent dimension table. For example, joining `PROD_SCD1_WID` from a sales table with `SCD1_WID` on `W_PRODUCT_D` will receive all instances of that product's data throughout history, even if the product has several different `ROW_WID` entries due to reclassifications, which insert new records to the dimension for the same item.

The other core structure to understand is Date WIDs (key column `DT_WID`). These also join with the `ROW_WID` of the parent dimension (`W_MCAL_DAY_D` usually), but the format of the WID allows you to extract the date value directly if needed, without table joins. The standard `DT_WID` value used is a combination of `1` + date in YYYYMMDD + `000`. For example, `120210815000` is the `DT_WID` value for "August 15, 2021".

Fact and Dimension Relationships

When a fact interface such as `SALES.csv` is loaded into the data warehouse, the programs perform many joins between the incoming data and the associated dimensions and hierarchies. The data warehouse enforces strict data integrity rules across tables; this requires that, for every fact record, all associated dimension records are present and active. For a record to be considered the active dimension record, it must have effective dates that encompass the same date on the fact record. For example:

- A record in the `SALES.csv` file for a given nightly batch cycle has key values of:
  - ITEM = 1285001
  - ORG_NUM = 4400
  - DAY_DT = 05-MAY-23
- For this record to be loaded successfully, the item, location, and date must all be present and active within the associated dimension tables:
  - For the item, a record must exist in `W_PRODUCT_D_RTL_TMP` having a matching item number in column `PROD_IT_NUM`, and the value of `DAY_DT` from the sale must be between `SRC_EFF_FROM_DT` and `SRC_EFF_TO_DT`.
  - For the location, a record must exist in `W_INT_ORG_DH_RTL_TMP` having a matching location number in column `ORG_NUM`, and the value of `DAY_DT` from the sale must be between `EFFECTIVE_FROM_DT` and `EFFECTIVE_TO_DT`.
  - For the date, a record must exist in `W_MCAL_DAY_D` having the same date on `MCAL_DAY_DT`.
- If the fact record has other key columns such as suppliers, buyers, reasons, statuses, and so on, then similar dimension joins may be used across all of them.

If any join between the fact table and dimension tables is unable to find matches based on the above criteria, then the fact record will not load into the data warehouse and will instead be rejected and moved into separate tables for review.

Transformations from Data Warehouse to Planning

Data Filtering and Conversions

In addition to simply aggregating the incoming fact data from item/location/date to item/location/week level, it is also important to understand what the data warehouse is doing with the data as it moves from the input files to the outbound interfaces. The table below summarizes the transformations and business logic applied to shared data warehouse facts used by RAP applications.

Table 6-2 Foundation Data Transformations

Transformation	Explanation
Currency Conversion	As part of the nightly batch, AIF DATA jobs will use exchange rate information to convert all incoming data from the source currency to the primary business currency. All data sent to downstream applications is in the primary currency. The data model maintains separate columns for both local and primary currency amounts for RI and AIF usage.
Tax Handling	The data model includes non-US taxes, such as VAT, in the sales retail amounts based on the indicators set up in the source system (such as Sales Audit) and in the data extraction jobs (RDE). When sending the sales data to Planning and AI Foundation, the default sales values may include VAT and only specific VAT-exclusive fields will remove it. You may optionally remove VAT from all data using configuration changes.
Transaction Date Usage	All fact data coming into the system includes a transaction date on the record. AIF DATA jobs aggregate from day to week level using transaction dates and does not alter or re-assign any records to different dates from what is provided. Transaction data in the past will be added to their historical week in the aggregates, no matter how far back it is dated.
Pack Item Handling	Downstream applications are currently only interested in the component item level, so AIF DATA will not send any fact data for pack items to other applications. Pack item sales must be spread to the component item level and loaded into the Sales Pack interface if this data is required for AI Foundation or Planning. All inventory, purchase order, and transaction data must be loaded at the component item level only.
Stockholding Locations	Inventory data for Planning is only exported for stockholding locations. A store indicated as a non-stockholding location on the location dimension will not be included in outbound inventory data. Physical warehouses which are not stockholding (because you use virtual warehouses) will also not be included.
Warehouse Types	Planning solutions assume that virtual warehouses are used as the stockholding locations for the business, and physical warehouses will be non-stockholding. For this reason, virtual warehouses are used to integrate data from the data warehouse to Planning, and no data is sent for the physical warehouses (except to indicate on each virtual WH the ID and name of the associated physical WH). If you don’t use virtual WHs, you can mark your physical WHs as virtual for the purposes of integration.

Table 6-2 (Cont.) Foundation Data Transformations

Transformation	Explanation
Future On Order	Planning applications require a forward-looking view of purchase orders based on the OTB EOW Date. The data warehouse accepts the actual purchase order details on the interfaces but will then transform the on-order amounts to be future-dated using the provided OTB EOW Dates. Orders which are past the OTB date will be included in the first EOW date, they will never be in the past.
Include On Order	Purchase Order data is limited by the Include On Order Flag on the Order Head interface. A value of `N` will not be included in the calculations for Planning.
Orderable Items	Purchase Order data is limited by the Orderable Flag on the Product interface. A value of `N` will not be included in the calculations for Planning.
Sellable Items	Regular non-pack items must be flagged as sellable to be interfaced to Planning as they do not want non-sellable item data in PDS at this time. This does not apply to pack items, which may be sellable or non-sellable because non-sellable pack items are often used for replenishment in IPO.
Inventory Adjustment Types	The system accepts 3 types of inventory adjustments using the codes `22`, `23`, and `41`. For Planning, only the first two codes are exported. Code `22` relates to Shrink and code `23` relates to Non-Shrink.
Inventory Receipt Types	The system accepts 3 types of inventory receipts using the codes `20`, `44~T`, and `44~A`. For Planning, all codes are sent but the `44`s are summed together. Code `20` relates to purchase order receipts. Code `44` relates to Transfer receipts and Allocation receipts. Only code `20` is used by MFP in the GA solution.
Inventory Transfer Types	The system accepts 3 types of transfers using the codes `N`, `B`, and `I` (normal, book, and intercompany). All three types are sent to planning along with the type codes.

Data Mappings

When you are generating input files to RAP, you may also want to know which columns are being moved to the output and how that data translates from what you see in the file to what you see in Planning applications. The list of mappings below describes how the data in the foundation data warehouse is exported to PDS.

> **Note:**
>
> Conversions and filters listed in the prior section of this chapter apply to all of this data (for example, data may be stored in local currency in RI but is always converted to the primary currency for export).

Product Mapping

The item dimension and product hierarchy data is loaded mainly from the `PRODUCT.csv` file or from RMFCS. The primary data warehouse table for item data is `W_PRODUCT_D` while the hierarchy comes from `W_PROD_CAT_DH`, but several temporary tables are used to pre-calculate the values before export. The mapping below is used by the interface program to move data from the data warehouse to RDX. The temporary table `W_RTL_ITEM_PARENT_TMP` is generated using data from `W_PROD_CAT_DH`, `W_PRODUCT_ATTR_D`, `W_PRODUCT_D_TL`, `W_RTL_IT_SUPPLIER_D`, and `W_DOMAIN_MEMBER_LKP_TL`. The export filters out non-pack items that have `SELLABLE_FLG=N` on the interface file or from Merchandising.

There is a configuration that alters the behavior of the `ITEM_DESC`, `ITEM_PARENT_DIFF_DESC`, and `ITEM_PARENT_DESC` fields. You may optionally update the `C_ODI_PARAM_VW` parameter `PDS_PROD_INCLUDE_ITEM_ID` to `Y`. When you do, the item ID will be concatenated into the description field on `W_PDS_PRODUCT_D` for all 3 levels of item.

There is also a configuration that alters the behavior of the `ITEM_PARENT_DIFF` field. You may optionally update the `C_ODI_PARAM_VW` parameter `ITEM_PARENT_DIFF_SEPARATOR` to any character (or to a null value). When you do, the concatenation of the ID values will use the given separator value instead of the default underscore character.

Measure	Target Table	Target Column	Data Source
Item	W_PDS_PRODUCT_D	ITEM	W_RTL_ITEM_PARENT_TMP.PROD_IT_NUM
Item Desc	W_PDS_PRODUCT_D	ITEM_DESC	W_RTL_ITEM_PARENT_TMP.PRODUCT_NAME
Item Parent Diff	W_PDS_PRODUCT_D	ITEM_PARENT_DIFF	CASE WHEN W_RTL_ITEM_PARENT_TMP.DIFF_AGGREGATE_ID = "-1" THEN W_RTL_ITEM_PARENT_TMP.PARENT_PROD_NUM ELSE W_RTL_ITEM_PARENT_TMP.PARENT_PROD_NUM||'${ ITEM_PARENT_DIFF_SEPARATOR}|| W_RTL_ITEM_PARENT_TMP.DIFF_AGGREGATE_ID END
Item Parent Diff Desc	W_PDS_PRODUCT_D	ITEM_PARENT_DIFF_DESC	CASE WHEN W_RTL_ITEM_PARENT_TMP.DIFF_AGGREGATE_ID = "-1" THEN W_RTL_ITEM_PARENT_TMP.PARENT_PRODUCT_NAME ELSE W_RTL_ITEM_PARENT_TMP.PARENT_PRODUCT_NAME||"_"|| NVL(W_RTL_ITEM_PARENT_TMP.DIFF_AGGREGATE_DESC , W_RTL_ITEM_PARENT_TMP.DIFF_AGGREGATE_ID) END
Item Parent	W_PDS_PRODUCT_D	ITEM_PARENT	W_RTL_ITEM_PARENT_TMP.PARENT_PROD_NUM
Item Parent Desc	W_PDS_PRODUCT_D	ITEM_PARENT_DESC	W_RTL_ITEM_PARENT_TMP.PARENT_PRODUCT_NAME
Subclass ID	W_PDS_PRODUCT_D	SUBCLASS_ID	W_RTL_ITEM_PARENT_TMP.SUBCLASS_ID

Measure	Target Table	Target Column	Data Source
Subclass Label	W_PDS_PRODUCT_D	SUB_NAME	W_RTL_ITEM_PARENT_TMP.PROD_SC_NUM||" "|| W_RTL_ITEM_PARENT_TMP.SBC_DESC
Class ID	W_PDS_PRODUCT_D	CLASS_ID	W_RTL_ITEM_PARENT_TMP.CLASS_ID
Class Label	W_PDS_PRODUCT_D	CLASS_NAME	W_RTL_ITEM_PARENT_TMP.PROD_CL_NUM||" "|| W_RTL_ITEM_PARENT_TMP.CLS_DESC
Department	W_PDS_PRODUCT_D	DEPT	W_RTL_ITEM_PARENT_TMP.PROD_DP_NUM
Department Label	W_PDS_PRODUCT_D	DEPT_NAME	W_RTL_ITEM_PARENT_TMP.PROD_DP_NUM||" "|| W_RTL_ITEM_PARENT_TMP.DP_DESC
Group	W_PDS_PRODUCT_D	GROUP_NO	W_RTL_ITEM_PARENT_TMP.PROD_GP_NUM
Group Label	W_PDS_PRODUCT_D	GROUP_NAME	W_RTL_ITEM_PARENT_TMP.PROD_GP_NUM||" "|| W_RTL_ITEM_PARENT_TMP.GRP_DESC
Division	W_PDS_PRODUCT_D	DIVISION	W_RTL_ITEM_PARENT_TMP.PROD_DV_NUM
Division Label	W_PDS_PRODUCT_D	DIV_NAME	W_RTL_ITEM_PARENT_TMP.PROD_DV_NUM||" "|| W_RTL_ITEM_PARENT_TMP.DIV_DESC
Company	W_PDS_PRODUCT_D	COMPANY	W_RTL_ITEM_PARENT_TMP.CMP_NUM
Company Label	W_PDS_PRODUCT_D	CO_NAME	W_RTL_ITEM_PARENT_TMP.CMP_NUM||" "|| W_RTL_ITEM_PARENT_TMP.CMP_DESC
Forecastable Flag	W_PDS_PRODUCT_D	FORECAST_IND	W_RTL_ITEM_PARENT_TMP.FORECAST_IND
Class	W_PDS_PRODUCT_D	CLASS_DISPLAY_ID	W_RTL_ITEM_PARENT_TMP.PROD_CL_NUM
Subclass	W_PDS_PRODUCT_D	SUBCLASS_DISPLAY_ID	W_RTL_ITEM_PARENT_TMP.PROD_SC_NUM
Brand	W_PDS_PRODUCT_D	BRAND_NAME	W_RTL_ITEM_PARENT_TMP.BRAND
Brand Label	W_PDS_PRODUCT_D	BRAND_DESCRIPTION	W_RTL_ITEM_PARENT_TMP.BRAND_DESC
Supplier	W_PDS_PRODUCT_D	SUPPLIER	NVL(W_PARTY_ORG_D.SUPPLIER_NUM"-1")
Supplier Label	W_PDS_PRODUCT_D	SUP_NAME	NVL(W_PARTY_ORG_D.ORG_NAME"N/A")
Diff 1	W_PDS_PRODUCT_D	DIFF_1	W_PRODUCT_D.DIFF_1

Measure	Target Table	Target Column	Data Source
Diff 1 Type	W_PDS_PRODUCT_D	DIFF_1_TYPE	W_PRODUCT_D.DIFF_1_TYPE
Diff 2	W_PDS_PRODUCT_D	DIFF_2	W_PRODUCT_D.DIFF_2
Diff 2 Type	W_PDS_PRODUCT_D	DIFF_2_TYPE	W_PRODUCT_D.DIFF_2_TYPE
Diff 3	W_PDS_PRODUCT_D	DIFF_3	W_PRODUCT_D.DIFF_3
Diff 3 Type	W_PDS_PRODUCT_D	DIFF_3_TYPE	W_PRODUCT_D.DIFF_3_TYPE
Diff 4	W_PDS_PRODUCT_D	DIFF_4	W_PRODUCT_D.DIFF_4
Diff 4 Type	W_PDS_PRODUCT_D	DIFF_4_TYPE	W_PRODUCT_D.DIFF_4_TYPE
Cost Zone Group ID	W_PDS_PRODUCT_D	COST_ZONE_GROUP_ID	W_PRODUCT_D.COST_ZONE_GROUP_ID
UOM Conv Factor	W_PDS_PRODUCT_D	UOM_CONV_FACTOR	W_PRODUCT_D.UOM_CONV_FACTOR
Store Order Multiple	W_PDS_PRODUCT_D	STORE_ORD_MULT	W_PRODUCT_D.STORE_ORD_MULT
Retail Label Type	W_PDS_PRODUCT_D	RETAIL_LABEL_TYPE	W_PRODUCT_D.RETAIL_LABEL_TYPE
Retail Label Value	W_PDS_PRODUCT_D	RETAIL_LABEL_VALUE	W_PRODUCT_D.RETAIL_LABEL_VALUE
Handling Temp	W_PDS_PRODUCT_D	HANDLING_TEMP	W_PRODUCT_D.HANDLING_TEMP
Handling Sensitivity	W_PDS_PRODUCT_D	HANDLING_SENSITIVITY	W_PRODUCT_D.HANDLING_SENSITIVITY
Catch Weight Flag	W_PDS_PRODUCT_D	CATCH_WEIGHT_IND	W_PRODUCT_D.CATCH_WEIGHT_IND
Waste Type	W_PDS_PRODUCT_D	WASTE_TYPE	W_PRODUCT_D.WASTE_TYPE
Waste Percent	W_PDS_PRODUCT_D	WASTE_PCT	W_PRODUCT_D.WASTE_PCT
Default Waste Percent	W_PDS_PRODUCT_D	DEFAULT_WASTE_PCT	W_PRODUCT_D.DEFAULT_WASTE_PCT
Item Service Level	W_PDS_PRODUCT_D	ITEM_SERVICE_LEVEL	W_PRODUCT_D.ITEM_SERVICE_LEVEL
Gift Wrap Flag	W_PDS_PRODUCT_D	GIFT_WRAP_IND	W_PRODUCT_D.GIFT_WRAP_IND
Ship Alone Flag	W_PDS_PRODUCT_D	SHIP_ALONE_IND	W_PRODUCT_D.SHIP_ALONE_IND
Order Type	W_PDS_PRODUCT_D	ORDER_TYPE	W_PRODUCT_D.ORDER_TYPE
Sales Type	W_PDS_PRODUCT_D	SALE_TYPE	W_PRODUCT_D.SALE_TYPE
Deposit Item Type	W_PDS_PRODUCT_D	DEPOSIT_ITEM_TYPE	W_PRODUCT_D.DEPOSIT_ITEM_TYPE

Measure	Target Table	Target Column	Data Source
Container Item	W_PDS_PRODUCT_D	CONTAINER_ITEM	W_PRODUCT_D.CONTAINER_ITEM
Deposit Price Per UOM	W_PDS_PRODUCT_D	DEPOSIT_IN_PRICE_PER_UOM	W_PRODUCT_D.DEPOSIT_IN_PRICE_PER_UOM
AIP Case Type	W_PDS_PRODUCT_D	AIP_CASE_TYPE	W_PRODUCT_D.AIP_CASE_TYPE
Perishable Flag	W_PDS_PRODUCT_D	PERISHABLE_IND	W_PRODUCT_D.PERISHABLE_IND
Catch Weight UOM	W_PDS_PRODUCT_D	CATCH_WEIGHT_UOM	W_PRODUCT_D.CATCH_WEIGHT_UOM
Orderable Flag	W_PDS_PRODUCT_D	ORDERABLE_FLG	W_PRODUCT_D.ORDERABLE_FLG
Inventoried Flag	W_PDS_PRODUCT_D	INVENTORIED_FLG	W_RTL_ITEM_PARENT_TMP.INVENTORIED_FLG
Flexible Attribute 1	W_PDS_PRODUCT_D	FLEX1_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX1_CHAR_VALUE, W_PRODUCT_D.FLEX1_CHAR_VALUE)
Flexible Attribute 2	W_PDS_PRODUCT_D	FLEX2_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX2_CHAR_VALUE, W_PRODUCT_D.FLEX2_CHAR_VALUE)
Flexible Attribute 3	W_PDS_PRODUCT_D	FLEX3_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX3_CHAR_VALUE, W_PRODUCT_D.FLEX3_CHAR_VALUE)
Flexible Attribute 4	W_PDS_PRODUCT_D	FLEX4_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX4_CHAR_VALUE, W_PRODUCT_D.FLEX4_CHAR_VALUE)
Flexible Attribute 5	W_PDS_PRODUCT_D	FLEX5_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX5_CHAR_VALUE, W_PRODUCT_D.FLEX5_CHAR_VALUE)
Flexible Attribute 6	W_PDS_PRODUCT_D	FLEX6_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX6_CHAR_VALUE, W_PRODUCT_D.FLEX6_CHAR_VALUE)
Flexible Attribute 7	W_PDS_PRODUCT_D	FLEX7_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX7_CHAR_VALUE, W_PRODUCT_D.FLEX7_CHAR_VALUE)
Flexible Attribute 8	W_PDS_PRODUCT_D	FLEX8_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX8_CHAR_VALUE, W_PRODUCT_D.FLEX8_CHAR_VALUE)
Flexible Attribute 9	W_PDS_PRODUCT_D	FLEX9_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX9_CHAR_VALUE, W_PRODUCT_D.FLEX9_CHAR_VALUE)

Measure	Target Table	Target Column	Data Source
Flexible Attribute 10	W_PDS_PRODUCT_D	FLEX10_CHAR_VALUE	COALESCE(W_PRODUCT_FLEX_D.FLEX10_CHAR_VALUE, W_PRODUCT_D.FLEX10_CHAR_VALUE)
Flexible Attribute 11	W_PDS_PRODUCT_D	FLEX11_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX11_CHAR_VALUE
Flexible Attribute 12	W_PDS_PRODUCT_D	FLEX12_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX12_CHAR_VALUE
Flexible Attribute 13	W_PDS_PRODUCT_D	FLEX13_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX13_CHAR_VALUE
Flexible Attribute 14	W_PDS_PRODUCT_D	FLEX14_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX14_CHAR_VALUE
Flexible Attribute 15	W_PDS_PRODUCT_D	FLEX15_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX15_CHAR_VALUE
Flexible Attribute 16	W_PDS_PRODUCT_D	FLEX16_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX16_CHAR_VALUE
Flexible Attribute 17	W_PDS_PRODUCT_D	FLEX17_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX17_CHAR_VALUE
Flexible Attribute 18	W_PDS_PRODUCT_D	FLEX18_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX18_CHAR_VALUE
Flexible Attribute 19	W_PDS_PRODUCT_D	FLEX19_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX19_CHAR_VALUE
Flexible Attribute 20	W_PDS_PRODUCT_D	FLEX20_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX20_CHAR_VALUE
Flexible Attribute 21	W_PDS_PRODUCT_D	FLEX21_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX21_CHAR_VALUE
Flexible Attribute 22	W_PDS_PRODUCT_D	FLEX22_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX22_CHAR_VALUE
Flexible Attribute 23	W_PDS_PRODUCT_D	FLEX23_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX23_CHAR_VALUE
Flexible Attribute 24	W_PDS_PRODUCT_D	FLEX24_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX24_CHAR_VALUE
Flexible Attribute 25	W_PDS_PRODUCT_D	FLEX25_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX25_CHAR_VALUE
Flexible Attribute 26	W_PDS_PRODUCT_D	FLEX26_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX26_CHAR_VALUE
Flexible Attribute 27	W_PDS_PRODUCT_D	FLEX27_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX27_CHAR_VALUE
Flexible Attribute 28	W_PDS_PRODUCT_D	FLEX28_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX28_CHAR_VALUE
Flexible Attribute 29	W_PDS_PRODUCT_D	FLEX29_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX29_CHAR_VALUE
Flexible Attribute 30	W_PDS_PRODUCT_D	FLEX30_CHAR_VALUE	W_PRODUCT_FLEX_D.FLEX30_CHAR_VALUE
Pack Flag	W_PDS_PRODUCT_D	PACK_FLG	W_PRODUCT_D.PACK_FLG
Image Filename	W_PDS_PRODUCT_D	PRODUCT_IMAGE_NAME	W_PRODUCT_ATTR_D.PRODUCT_IMAGE_NAME

Measure	Target Table	Target Column	Data Source
Image Address	W_PDS_PRODUCT_D	PRODUCT_IMAGE_ADDR	W_PRODUCT_ATTR_D.PRODUCT_IMAGE_ADDR
Attr Image Filename	W_PDS_PRODUCT_D	PRODUCT_ATTR_IMAGE_NAME	W_PRODUCT_ATTR_D.PRODUCT_ATTR_IMAGE_NAME
Attr Image Address	W_PDS_PRODUCT_D	PRODUCT_ATTR_IMAGE_ADDR	W_PRODUCT_ATTR_D.PRODUCT_ATTR_IMAGE_ADDR
Image URL	W_PDS_PRODUCT_D	PRODUCT_IMAGE	W_PRODUCT_ATTR_D.PRODUCT_IMAGE_ADDR || W_PRODUCT_ATTR_D.PRODUCT_IMAGE_NAME
Parent Image Filename	W_PDS_PRODUCT_D	ITEM_PARENT_IMAGE_NAME	W_PRODUCT_ATTR_D.PRODUCT_IMAGE_NAME (for parent style item)
Parent Image Address	W_PDS_PRODUCT_D	ITEM_PARENT_ADDR	W_PRODUCT_ATTR_D.PRODUCT_IMAGE_ADDR (for parent style item)
Parent Image URL	W_PDS_PRODUCT_D	ITEM_PARENT_IMAGE	W_PRODUCT_ATTR_D.PRODUCT_IMAGE_ADDR || W_PRODUCT_ATTR_D.PRODUCT_IMAGE_NAME (for parent style item)
Sellable Flag	W_PDS_PRODUCT_D	SELLABLE_FLG	W_PRODUCT_ATTR_D.PRODUCT_ATTR4_NAME
Initial Item Cost	W_PDS_PRODUCT_D	INITIAL_ITEM_COST	W_PRODUCT_ATTR_D.PRODUCT_ATTR1_NUM_VALUE
Initial Item Retail	W_PDS_PRODUCT_D	INITIAL_ITEM_RETAIL	W_PRODUCT_ATTR_D.PRODUCT_ATTR5_NUM_VALUE

Organization Mapping

The location dimension and organization hierarchy data is loaded mainly from the `ORGANIZATION.csv` file or from RMFCS. The primary data warehouse table for location data is `W_INT_ORG_D` while the hierarchy comes from `W_INT_ORG_DH`, but several other tables are used to pre-calculate the values before export. The mapping below is used by the interface program to move data from the data warehouse to RDX. `W_DOMAIN_MEMBER_LKP_TL` is the holding table for translatable description strings. `W_INT_ORG_ATTR_D` is for location attributes. Other tables ending in `TL` are for lookup strings for specific entities like store names. The mappings are separated by store and warehouse, when different logic is used based on the location type. Only virtual warehouses are exported here, physical warehouse records are excluded from the export.

Measure	Target Table	Target Column	Data Source
Location	W_PDS_ORGANIZATION_D	LOCATION	W_INT_ORG_D.ORG_NUM
Location Label	W_PDS_ORGANIZATION_D	LOC_NAME	W_INT_ORG_D.ORG_NUM||" "|| W_INT_ORG_D_TL.ORG_NAME
District	W_PDS_ORGANIZATION_D	DISTRICT	'WH'||W_INT_ORG_D.ORG_NUM (warehouses), W_INT_ORG_DH_RTL_TMP.ORG_DS_NUM (stores)

Measure	Target Table	Target Column	Data Source
District Label	W_PDS_ORGANIZATION_D	DISTRICT_NAME	W_INT_ORG_D_TL.ORG_NAME (warehouses), W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME (stores)
Region	W_PDS_ORGANIZATION_D	REGION	"WH"||W_INT_ORG_D.ORG_NUM (warehouses), W_INT_ORG_DH_RTL_TMP.ORG_RG_NUM (stores)
Region Label	W_PDS_ORGANIZATION_D	REGION_NAME	W_INT_ORG_D_TL.ORG_NAME (warehouses), W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME (stores)
Area	W_PDS_ORGANIZATION_D	AREA	"WH"||W_INT_ORG_D.ORG_NUM (warehouses), W_INT_ORG_DH_RTL_TMP.ORG_AR_NUM (stores)
Area Label	W_PDS_ORGANIZATION_D	AREA_NAME	W_INT_ORG_D_TL.ORG_NAME (warehouses), W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME (stores)
Chain	W_PDS_ORGANIZATION_D	CHAIN	"WH"||W_INT_ORG_D.ORG_NUM (warehouses), W_INT_ORG_DH_RTL_TMP.ORG_CH_NUM (stores)
Chain Label	W_PDS_ORGANIZATION_D	CHAIN_NAME	W_INT_ORG_D_TL.ORG_NAME (warehouses), W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME (stores)
Company	W_PDS_ORGANIZATION_D	COMPANY	W_INT_ORG_DH.ORG_TOP_NUM
Company Label	W_PDS_ORGANIZATION_D	CO_NAME	W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME
Company Primary Currency	W_PDS_ORGANIZATION_D	COMPANY_CURRENCY	RA_SRC_CURR_PARAM_G.COMPANY_CURRENCY
Location Type Code	W_PDS_ORGANIZATION_D	LOC_TYPE	W_INT_ORG_ATTR_D.ORG_ATTR42_NAME
Location Type	W_PDS_ORGANIZATION_D	LOC_TYPE_NAME	"Warehouse" or "Store"
Physical Warehouse ID for VWH	W_PDS_ORGANIZATION_D	PHYSICAL_WH	W_INT_ORG_ATTR_D.ORG_ATTR14_NAME
Physical Warehouse Name	W_PDS_ORGANIZATION_D	PHYSICAL_WH_NAME	W_INT_ORG_D_TL.ORG_NAME
Channel ID	W_PDS_ORGANIZATION_D	CHANNEL_ID	NVL(TO_CHAR(W_INT_ORG_ATTR_D.ORG_ATTR5_NUM_VALUE) "NA")
Channel Desc	W_PDS_ORGANIZATION_D	CHANNEL_NAME	NVL(W_INT_ORG_ATTR_D.ORG_ATTR5_NAME "UNASSIGNED")

Measure	Target Table	Target Column	Data Source
Store Class	W_PDS_ORGANIZATION_D	STORE_CLASS	CASE NVL(W_INT_ORG_ATTR_D.ORG_ATTR41_NAME,''-1'') WHEN ''-1'' THEN ''NA'' ELSE W_INT_ORG_ATTR_D.ORG_ATTR41_NAME END
Store Class Desc	W_PDS_ORGANIZATION_D	STORE_CLASS_DESCRIPTION	CASE NVL(W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME,''-1'') WHEN ''-1'' THEN ''NA'' ELSE W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME END
Store Format	W_PDS_ORGANIZATION_D	STORE_FORMAT	CASE NVL(W_INT_ORG_ATTR_D.ORG_ATTR22_NAME,''-1'') WHEN ''-1'' THEN ''NA'' ELSE W_INT_ORG_ATTR_D.ORG_ATTR22_NAME END
Store Format Desc	W_PDS_ORGANIZATION_D	STORE_FORMAT_NAME	CASE NVL(W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME,''-1'') WHEN ''-1'' THEN ''NA'' ELSE W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME END
Store Close Date	W_PDS_ORGANIZATION_D	STORE_CLOSE_DATE	W_INT_ORG_ATTR_D.ORG_ATTR3_DATE
Store Open Date	W_PDS_ORGANIZATION_D	STORE_OPEN_DATE	W_INT_ORG_ATTR_D.ORG_ATTR2_DATE
Store Remodel Date	W_PDS_ORGANIZATION_D	REMODEL_DATE	W_INT_ORG_ATTR_D.ORG_ATTR1_DATE
Location Currency Code	W_PDS_ORGANIZATION_D	CURRENCY	W_INT_ORG_D.W_CURR_CODE
Store Type	W_PDS_ORGANIZATION_D	STORE_TYPE	W_INT_ORG_ATTR_D.ORG_ATTR23_NAME
Stockholding Flag	W_PDS_ORGANIZATION_D	STOCKHOLDING_IND	NVL(W_INT_ORG_ATTR_D.ORG_ATTR19_NAME, 'Y')
Default Warehouse ID	W_PDS_ORGANIZATION_D	DEFAULT_WH_ID	W_INT_ORG_ATTR_D.ORG_ATTR20_NAME
Store Format Description	W_PDS_ORGANIZATION_D	STORE_FORMAT_DESC	W_INT_ORG_ATTR_D.ORG_ATTR21_NAME
Store Format ID	W_PDS_ORGANIZATION_D	STORE_FORMAT_ID	W_INT_ORG_ATTR_D.ORG_ATTR22_NAME
Store UPS Disst	W_PDS_ORGANIZATION_D	STORE_UPS_DIST	W_INT_ORG_ATTR_D.ORG_ATTR24_NAME
Time Zone	W_PDS_ORGANIZATION_D	TIME_ZONE	W_INT_ORG_ATTR_D.ORG_ATTR25_NAME
Transfer Zone ID	W_PDS_ORGANIZATION_D	TRANSFER_ZONE_ID	W_INT_ORG_ATTR_D.ORG_ATTR26_NAME
Transfer Zone Description	W_PDS_ORGANIZATION_D	TRANSFER_ZONE_DESC	W_INT_ORG_ATTR_D.ORG_ATTR27_NAME

Measure	Target Table	Target Column	Data Source
VAT Region ID	W_PDS_ORGANIZATION_D	VAT_REGION_ID	W_INT_ORG_ATTR_D.ORG_ATTR28_NAME
VAT Include Flag	W_PDS_ORGANIZATION_D	VAT_INCLUDE_FLG	W_INT_ORG_ATTR_D.ORG_ATTR29_NAME
Virtual Warehouse Flag	W_PDS_ORGANIZATION_D	VIRTUAL_WH_FLG	W_INT_ORG_ATTR_D.ORG_ATTR30_NAME
Transfer Entity ID	W_PDS_ORGANIZATION_D	TRANSFER_ENTITY_ID	W_INT_ORG_ATTR_D.ORG_ATTR31_NAME
Transfer Entity Description	W_PDS_ORGANIZATION_D	TRANSFER_ENTITY_DESC	W_INT_ORG_ATTR_D.ORG_ATTR32_NAME
Wholesale/ Franchise Cust Type	W_PDS_ORGANIZATION_D	WF_CUST_TYPE	W_INT_ORG_ATTR_D.ORG_ATTR35_NAME
Wholesale/ Franchise Group ID	W_PDS_ORGANIZATION_D	WF_GROUP_ID	W_INT_ORG_ATTR_D.ORG_ATTR36_NAME
Wholesale/ Franchise Group Name	W_PDS_ORGANIZATION_D	WF_GROUP_NAME	W_INT_ORG_ATTR_D.ORG_ATTR37_NAME
Wholesale/ Franchise Cust ID	W_PDS_ORGANIZATION_D	WF_CUST_ID	W_INT_ORG_ATTR_D.ORG_ATTR38_NAME
Wholesale/ Franchise Cust Name	W_PDS_ORGANIZATION_D	WF_CUST_NAME	W_INT_ORG_ATTR_D.ORG_ATTR39_NAME
Sister Store ID	W_PDS_ORGANIZATION_D	SISTER_STORE_ID	W_INT_ORG_ATTR_D.ORG_ATTR40_NAME
Store Class Type	W_PDS_ORGANIZATION_D	STORE_CLASS_TYPE	W_INT_ORG_ATTR_D.ORG_ATTR41_NAME
Store Class Desc	W_PDS_ORGANIZATION_D	STORE_CLASS_DESC	W_INT_ORG_ATTR_D.ORG_ATTR44_NAME
Customer Order Location Indicator	W_PDS_ORGANIZATION_D	CUST_ORDER_LOC_IND	W_INT_ORG_ATTR_D.ORG_ATTR48_NAME
Customer Order Shipping Indicator	W_PDS_ORGANIZATION_D	CUST_ORDER_SHIP_IND	W_INT_ORG_ATTR_D.ORG_ATTR49_NAME
Gift Wrapping Indicator	W_PDS_ORGANIZATION_D	GIFT_WRAPPING_IND	W_INT_ORG_ATTR_D.ORG_ATTR50_NAME
Location Language ISO Code	W_PDS_ORGANIZATION_D	LANG_ISO_CODE	W_INT_ORG_ATTR_D.ORG_ATTR51_NAME
WH Delivery Policy	W_PDS_ORGANIZATION_D	WH_DELIVERY_POLICY	W_INT_ORG_ATTR_D.ORG_ATTR54_NAME
WH Redistribution Indicator	W_PDS_ORGANIZATION_D	WH_REDIST_IND	W_INT_ORG_ATTR_D.ORG_ATTR55_NAME

Measure	Target Table	Target Column	Data Source
WH Replenishment Indicator	W_PDS_ORGANIZATION_D	WH_REPL_IND	W_INT_ORG_ATTR_D.ORG_ATTR56_NAME
WH Finisher Indicator	W_PDS_ORGANIZATION_D	WH_FINISHER_IND	W_INT_ORG_ATTR_D.ORG_ATTR57_NAME
Virtual WH Type	W_PDS_ORGANIZATION_D	VIRTUAL_WH_TYPE	W_INT_ORG_ATTR_D.ORG_ATTR58_NAME
DUNS Number	W_PDS_ORGANIZATION_D	DUNS_NUMBER	W_INT_ORG_ATTR_D.ORG_ATTR59_NAME
DUNS Location	W_PDS_ORGANIZATION_D	DUNS_LOC	W_INT_ORG_ATTR_D.ORG_ATTR60_NAME
Selling Area Sq. Ft.	W_PDS_ORGANIZATION_D	SELLING_AREA	W_INT_ORG_ATTR_D.ORG_ATTR1_NUM_VALUE
Linear Distance	W_PDS_ORGANIZATION_D	LINEAR_DISTANCE	W_INT_ORG_ATTR_D.ORG_ATTR2_NUM_VALUE
Total Sq. Ft.	W_PDS_ORGANIZATION_D	TOTAL_AREA	W_INT_ORG_ATTR_D.ORG_ATTR3_NUM_VALUE
WH Inbound Handling Days	W_PDS_ORGANIZATION_D	INBOUND_HANDLING_DAYS	W_INT_ORG_ATTR_D.ORG_ATTR6_NUM_VALUE
Stop Order Days	W_PDS_ORGANIZATION_D	STOP_ORDER_DAYS	W_INT_ORG_ATTR_D.ORG_ATTR7_NUM_VALUE
Start Order Days	W_PDS_ORGANIZATION_D	START_ORDER_DAYS	W_INT_ORG_ATTR_D.ORG_ATTR8_NUM_VALUE
Flexible Attribute 1	W_PDS_ORGANIZATION_D	FLEX1_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX1_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX1_CHAR_VALUE)
Flexible Attribute 2	W_PDS_ORGANIZATION_D	FLEX2_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX2_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX2_CHAR_VALUE)
Flexible Attribute 3	W_PDS_ORGANIZATION_D	FLEX3_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX3_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX3_CHAR_VALUE)
Flexible Attribute 4	W_PDS_ORGANIZATION_D	FLEX4_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX4_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX4_CHAR_VALUE)
Flexible Attribute 5	W_PDS_ORGANIZATION_D	FLEX5_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX5_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX5_CHAR_VALUE)
Flexible Attribute 6	W_PDS_ORGANIZATION_D	FLEX6_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX6_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX6_CHAR_VALUE)
Flexible Attribute 7	W_PDS_ORGANIZATION_D	FLEX7_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX7_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX7_CHAR_VALUE)

Measure	Target Table	Target Column	Data Source
Flexible Attribute 8	W_PDS_ORGANIZATION_D	FLEX8_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX8_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX8_CHAR_VALUE)
Flexible Attribute 9	W_PDS_ORGANIZATION_D	FLEX9_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX9_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX9_CHAR_VALUE)
Flexible Attribute 10	W_PDS_ORGANIZATION_D	FLEX10_CHAR_VALUE	COALESCE(W_ORGANIZATION_FLEX_D.FLEX10_CHAR_VALUE,W_INT_ORG_ATTR_D.FLEX10_CHAR_VALUE)
Flexible Attribute 11	W_PDS_ORGANIZATION_D	FLEX11_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX11_CHAR_VALUE
Flexible Attribute 12	W_PDS_ORGANIZATION_D	FLEX12_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX12_CHAR_VALUE
Flexible Attribute 13	W_PDS_ORGANIZATION_D	FLEX13_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX13_CHAR_VALUE
Flexible Attribute 14	W_PDS_ORGANIZATION_D	FLEX14_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX14_CHAR_VALUE
Flexible Attribute 15	W_PDS_ORGANIZATION_D	FLEX15_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX15_CHAR_VALUE
Flexible Attribute 16	W_PDS_ORGANIZATION_D	FLEX16_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX16_CHAR_VALUE
Flexible Attribute 17	W_PDS_ORGANIZATION_D	FLEX17_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX17_CHAR_VALUE
Flexible Attribute 18	W_PDS_ORGANIZATION_D	FLEX18_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX18_CHAR_VALUE
Flexible Attribute 19	W_PDS_ORGANIZATION_D	FLEX19_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX19_CHAR_VALUE
Flexible Attribute 20	W_PDS_ORGANIZATION_D	FLEX20_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX20_CHAR_VALUE
Flexible Attribute 21	W_PDS_ORGANIZATION_D	FLEX21_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX21_CHAR_VALUE
Flexible Attribute 22	W_PDS_ORGANIZATION_D	FLEX22_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX22_CHAR_VALUE
Flexible Attribute 23	W_PDS_ORGANIZATION_D	FLEX23_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX23_CHAR_VALUE
Flexible Attribute 24	W_PDS_ORGANIZATION_D	FLEX24_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX24_CHAR_VALUE
Flexible Attribute 25	W_PDS_ORGANIZATION_D	FLEX25_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX25_CHAR_VALUE
Flexible Attribute 26	W_PDS_ORGANIZATION_D	FLEX26_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX26_CHAR_VALUE
Flexible Attribute 27	W_PDS_ORGANIZATION_D	FLEX27_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX27_CHAR_VALUE
Flexible Attribute 28	W_PDS_ORGANIZATION_D	FLEX28_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX28_CHAR_VALUE

Measure	Target Table	Target Column	Data Source
Flexible Attribute 29	W_PDS_ORGANIZATION_D	FLEX29_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX29_CHAR_VALUE
Flexible Attribute 30	W_PDS_ORGANIZATION_D	FLEX30_CHAR_VALUE	W_ORGANIZATION_FLEX_D.FLEX30_CHAR_VALUE

Calendar Mapping

The calendar hierarchy data is loaded from the `CALENDAR.csv` file or from RMFCS. The calendar must be a fiscal calendar (such as 4-4-5 or 4-5-4). The primary data warehouse table having day-level data is `W_MCAL_DAY_D`. AIF DATA jobs automatically generate the calendar using the start/end dates for the fiscal periods in the input file. AIF DATA jobs also generate an internal Gregorian calendar at the same time the fiscal calendar is loaded, and this data is exported alongside the fiscal calendar for extensions and customizations.

Measure	Target Table	Target Column	Data Source
Day Date	W_PDS_CALENDAR_D	DAY	W_MCAL_DAY_D.MCAL_DAY_DT
Week Ending Date	W_PDS_CALENDAR_D	WEEK	W_MCAL_DAY_D.MCAL_WEEK_END_DT
Month Number	W_PDS_CALENDAR_D	MONTH	W_MCAL_DAY_D.MCAL_PERIOD
Quarter Number	W_PDS_CALENDAR_D	QUARTER	W_MCAL_DAY_D.MCAL_QTR
Half Year Number	W_PDS_CALENDAR_D	HALF	CASE WHEN W_MCAL_DAY_D.MCAL_QTR <= 2 THEN 1 ELSE 2 END
Year Number	W_PDS_CALENDAR_D	YEAR	W_MCAL_DAY_D.MCAL_YEAR
Week of Year	W_PDS_CALENDAR_D	WEEK_OF_YEAR	W_MCAL_DAY_D.MCAL_WEEK_OF_YEAR
Day of Week	W_PDS_CALENDAR_D	DAY_OF_WEEK	W_MCAL_DAY_D.MCAL_DAY_OF_WEEK
Gregorian Day of Week	W_PDS_CALENDAR_D	GREG_DAY_OF_WEEK	W_MCAL_DAY_D.CAL_DAY_OF_THE_WEEK_CODE
Gregorian Day of Month	W_PDS_CALENDAR_D	GREG_DAY_OF_MONTH	W_MCAL_DAY_D.CAL_DAY_OF_MONTH
Gregorian Day of Year	W_PDS_CALENDAR_D	GREG_DAY_OF_YEAR	W_MCAL_DAY_D.CAL_DAY_OF_YEAR
Gregorian Week ID	W_PDS_CALENDAR_D	GREG_WEEK_ID	W_MCAL_DAY_D.CAL_WEEK_WID
Gregorian Week Ending Date	W_PDS_CALENDAR_D	GREG_WEEK_END_DT	W_MCAL_DAY_D.CAL_WEEK_END_DT
Gregorian Week Name	W_PDS_CALENDAR_D	GREG_WEEK_NAME	W_MCAL_DAY_D.CAL_WEEK_NAME
GregorianMonth ID	W_PDS_CALENDAR_D	GREG_MONTH_ID	W_MCAL_DAY_D.CAL_MONTH_WID

Measure	Target Table	Target Column	Data Source
Gregorian Month of Year	W_PDS_CALENDAR_D	GREG_MONTH	W_MCAL_DAY_D.CAL_MONTH
Gregorian Month Ending Date	W_PDS_CALENDAR_D	GREG_MONTH_END_DT	W_MCAL_DAY_D.CAL_MONTH_END_DT
Gregorian Month Name	W_PDS_CALENDAR_D	GREG_MONTH_NAME	W_MCAL_DAY_D.W_CAL_MONTH_CODE
Gregorian Quarter ID	W_PDS_CALENDAR_D	GREG_QTR_ID	W_MCAL_DAY_D.CAL_QTR_WID
Gregorian Quarter of Year	W_PDS_CALENDAR_D	GREG_QTR	W_MCAL_DAY_D.CAL_QTR
Gregorian Quarter Ending Date	W_PDS_CALENDAR_D	GREG_QTR_END_DT	W_MCAL_DAY_D.CAL_QTR_END_DT
Gregorian Quarter Name	W_PDS_CALENDAR_D	GREG_QTR_NAME	W_MCAL_DAY_D.CAL_QTR_NAME
Gregorian Half of Year	W_PDS_CALENDAR_D	GREG_HALF	W_MCAL_DAY_D.CAL_HALF
Gregorian Half Name	W_PDS_CALENDAR_D	GREG_HALF_NAME	W_MCAL_DAY_D.CAL_HALF_NAME
Gregorian Year	W_PDS_CALENDAR_D	GREG_YEAR	W_MCAL_DAY_D.CAL_YEAR
Gregorian Year Ending Date	W_PDS_CALENDAR_D	GREG_YEAR_END_DT	W_MCAL_DAY_D.CAL_YEAR_END_DT
Gregorian Half of Year ID	W_PDS_CALENDAR_D	GREG_HALF_ID	W_MCAL_DAY_D.CAL_YEAR || W_MCAL_DAY_D.CAL_HALF

Exchange Rate Mapping

The exchange rate data is loaded from the `EXCH_RATE.csv` file or from RMFCS.

Measure	Target Table	Target Column	Data Source
Start Date	W_PDS_EXCH_RATE_G	EFFECTIVE_DATE	W_EXCH_RATE_G.START_DT
From Currency Code	W_PDS_EXCH_RATE_G	FROM_CURRENCY_CODE	W_EXCH_RATE_G.W_FROM_CURCY_CODE
To Currency Code	W_PDS_EXCH_RATE_G	TO_CURRENCY_CODE	W_EXCH_RATE_G.W_TO_CURCY_CODE
Exchange Rate Type	W_PDS_EXCH_RATE_G	EXCHANGE_TYPE	W_EXCH_RATE_G.RATE_TYPE
Exchange Rate	W_PDS_EXCH_RATE_G	EXCHANGE_RATE	W_EXCH_RATE_G.EXCH_RATE

User Defined Attributes (UDA) Mapping

The user-defined attributes label data is loaded from the `ATTR.csv` file or from RMFCS. This table is only for the attribute group and value labels and hierarchy. UDA type code refers to 3 types (LV, FF, or DT) which is a list of values, free-form text, or date attribute type. Different implementations may require different subsets of UDAs from their source system.

Measure	Target Table	Target Column	Data Source
UDA Type	W_PDS_UDA_D	UDA_TYPE_CODE	W_RTL_PRODUCT_ATTR_D.PROD_ATTR_TYPE
UDA Group ID	W_PDS_UDA_D	UDA_ID	W_RTL_PRODUCT_ATTR_D.PROD_ATTR_GROUP_ID
UDA Group Desc	W_PDS_UDA_D	UDA_DESC	W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME
UDA Value ID	W_PDS_UDA_D	UDA_VALUE	W_RTL_PRODUCT_ATTR_D.PROD_ATTR_ID
UDA Value Desc	W_PDS_UDA_D	UDA_VALUE_DESC	W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME

Differentiator Attributes Mapping

The differentiator data is loaded from the ATTR.csv file or from RMFCS. This table is only for the attribute group and value labels and hierarchy. Diffs include any input data marked as type DIFF, as well as pre-defined diff types such as COLOR and SIZE.

Measure	Target Table	Target Column	Data Source
Diff Group ID	W_PDS_DIFF_D	DIFF_TYPE_ID	W_RTL_PRODUCT_ATTR_D.PROD_ATTR_GROUP_ID
Diff Group Desc	W_PDS_DIFF_D	DIFF_TYPE_DESC	W_RTL_PRODUCT_ATTR_D_TL.PROD_ATTR_TYPE_DESC
Diff ID	W_PDS_DIFF_D	DIFF_ID	W_RTL_PRODUCT_ATTR_D.PROD_ATTR_ID
Diff Desc	W_PDS_DIFF_D	DIFF_DESC	W_RTL_PRODUCT_ATTR_D_TL.PROD_ATTR_DESC

Item Attributes Mapping

The item attribute relationship data is loaded from the PROD_ATTR.csv file or from RMFCS. This is the relationship between items (SKUs) and their attributes (UDAs).

Measure	Target Table	Target Column	Data Source
Item	W_PDS_PRODUCT_ATTR_D	ITEM	W_PRODUCT_D_RTL_TMP.PROD_IT_NUM
UDA Type	W_PDS_PRODUCT_ATTR_D	UDA_TYPE	W_RTL_ITEM_GRP1_D.FLEX_ATTRIB_2_CHAR
UDA Group ID	W_PDS_PRODUCT_ATTR_D	UDA_ID	W_RTL_ITEM_GRP1_D.FLEX_ATTRIB_1_CHAR
UDA Group Desc	W_PDS_PRODUCT_ATTR_D	UDA_DESC	W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME
UDA Value ID	W_PDS_PRODUCT_ATTR_D	UDA_VALUE	W_RTL_ITEM_GRP1_D.FLEX_ATTRIB_3_CHAR
UDA Value Desc	W_PDS_PRODUCT_ATTR_D	UDA_VALUE_DESC	W_DOMAIN_MEMBER_LKP_TL.DOMAIN_MEMBER_NAME

Differentiator Group Mapping

The differentiator groups data is loaded from the `DIFF_GROUP.csv` file or from RMFCS. These are for assortment planning diff group hierarchy and are the same groups used in AIF Size Profile Science.

Measure	Target Table	Target Column	Data Source
Diff Type	W_PDS_DIFF_GRP_D	DIFF_TYPE_ID	W_RTL_DIFF_GRP_D. DIFF_TYPE
Diff ID	W_PDS_DIFF_GRP_D	DIFF_ID	W_RTL_DIFF_GRP_D. DIFF_ID
Diff Desc	W_PDS_DIFF_GRP_D	DIFF_DESC	W_DOMAIN_MEMBER_LKP_TL. DOMAIN_MEMBER_NAME
Diff Group ID	W_PDS_DIFF_GRP_D	DIFF_GROUP_ID	W_RTL_DIFF_GRP_D. DIFF_GROUP_ID
Diff Group Desc	W_PDS_DIFF_GRP_D	DIFF_GROUP_DESC	W_RTL_DIFF_GRP_D_TL. DIFF_GROUP_DESC

Brand Mapping

The brand data is loaded from the `PRODUCT.csv` file or from RMFCS. The product data load programs will insert the brand information into the additional tables used below (as long as these tables are enabled during foundation loads).

Measure	Target Table	Target Column	Data Source
Brand ID	W_PDS_BRAND_D	BRAND_NAME	W_RTL_PRODUCT_BRAND_D. BRAND_ID
Brand Desc	W_PDS_BRAND_D	BRAND_DESCRIPTION	W_RTL_PRODUCT_BRAND_D_TL. BRAND_DESCR

Replenishment Attribute Mapping

The replenishment attribute data is loaded from the `PROD_LOC_REPL.csv` file or from RMFCS. This data is not used by any planning solution in the default templates, but it is made available for customer extensions.

Measure	Target Table	Target Column	Data Source
Item	W_PDS_REPL_ATTR_IT_LC_D	ITEM	W_PRODUCT_D_RTL_TMP.PROD_IT_NUM
Location	W_PDS_REPL_ATTR_IT_LC_D	LOCATION	W_INT_ORG_D_RTL_TMP.ORG_NUM
Food Stamp Flag	W_PDS_REPL_ATTR_IT_LC_D	FOOD_STAMP_IND	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR1_NAME
Reward Eligible Flag	W_PDS_REPL_ATTR_IT_LC_D	REWARD_ELIGIBLE_IND	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR2_NAME
Natl Brand Comp Item	W_PDS_REPL_ATTR_IT_LC_D	NATL_BRAND_COMP_ITEM	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR3_NAME
Elect Mkt Clubs	W_PDS_REPL_ATTR_IT_LC_D	ELECT_MKT_CLUBS	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR4_NAME

Measure	Target Table	Target Column	Data Source
Store Reorderable Flag	W_PDS_REPL_ATTR_IT_LC_D	STORE_REORDERABLE_IND	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR5_NAME
Manual Price Entry	W_PDS_REPL_ATTR_IT_LC_D	MANUAL_PRICE_ENTRY	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR6_NAME
WIC Flag	W_PDS_REPL_ATTR_IT_LC_D	WIC_IND	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR7_NAME
In Store Market Basket	W_PDS_REPL_ATTR_IT_LC_D	IN_STORE_MARKET_BASKET	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR8_NAME
Returnable Flag	W_PDS_REPL_ATTR_IT_LC_D	RETURNABLE_IND	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR9_NAME
Launch Date	W_PDS_REPL_ATTR_IT_LC_D	LAUNCH_DATE	W_INVENTORY_PRODUCT_ATTR_D.INV_ATTR1_DATE
Refundable Flag	W_PDS_REPL_ATTR_IT_LC_D	REFUNDABLE_IND	W_INVENTORY_PRODUCT_ATTR_D.REFUNDABLE_IND
Back Order Flag	W_PDS_REPL_ATTR_IT_LC_D	BACK_ORDER_IND	W_INVENTORY_PRODUCT_ATTR_D.BACK_ORDER_IND
Replenishment Supplier Num	W_PDS_REPL_ATTR_IT_LC_D	REPL_SUPPLIER_NUM	W_INVENTORY_PRODUCT_ATTR_D.REPL_SUPPLIER_NUM
Replenishment Country Code	W_PDS_REPL_ATTR_IT_LC_D	REPL_COUNTRY_CODE	W_INVENTORY_PRODUCT_ATTR_D.REPL_COUNTRY_CODE
Replenishment Review Cycle	W_PDS_REPL_ATTR_IT_LC_D	REPL_REVIEW_CYCLE	W_INVENTORY_PRODUCT_ATTR_D.REPL_REVIEW_CYCLE
Replenishment Stock Cat	W_PDS_REPL_ATTR_IT_LC_D	REPL_STOCK_CAT	W_INVENTORY_PRODUCT_ATTR_D.REPL_STOCK_CAT
Replenishment Source Wh	W_PDS_REPL_ATTR_IT_LC_D	REPL_SOURCE_WH	W_INVENTORY_PRODUCT_ATTR_D.REPL_SOURCE_WH
Replenishment Activate Dt	W_PDS_REPL_ATTR_IT_LC_D	REPL_ACTIVATE_DT	W_INVENTORY_PRODUCT_ATTR_D.REPL_ACTIVATE_DT
Replenishment Deactivate Dt	W_PDS_REPL_ATTR_IT_LC_D	REPL_DEACTIVATE_DT	NVL(W_INVENTORY_PRODUCT_ATTR_D.REPL_DEACTIVATE_DT ,'2100-01-01')
Replenishment Pres Stock	W_PDS_REPL_ATTR_IT_LC_D	REPL_PRES_STOCK	W_INVENTORY_PRODUCT_ATTR_D.REPL_PRES_STOCK
Replenishment Demo Stock	W_PDS_REPL_ATTR_IT_LC_D	REPL_DEMO_STOCK	W_INVENTORY_PRODUCT_ATTR_D.REPL_DEMO_STOCK
Replenishment Min Stock	W_PDS_REPL_ATTR_IT_LC_D	REPL_MIN_STOCK	W_INVENTORY_PRODUCT_ATTR_D.REPL_MIN_STOCK
Replenishment Max Stock	W_PDS_REPL_ATTR_IT_LC_D	REPL_MAX_STOCK	W_INVENTORY_PRODUCT_ATTR_D.REPL_MAX_STOCK
Replenishment Service Level	W_PDS_REPL_ATTR_IT_LC_D	REPL_SERVICE_LEVEL	W_INVENTORY_PRODUCT_ATTR_D.REPL_SERVICE_LEVEL
Replenishment Pickup Leadtime	W_PDS_REPL_ATTR_IT_LC_D	REPL_PICKUP_LEADTIME	W_INVENTORY_PRODUCT_ATTR_D.REPL_PICKUP_LEADTIME
Replenishment Wh Leadtime	W_PDS_REPL_ATTR_IT_LC_D	REPL_WH_LEADTIME	W_INVENTORY_PRODUCT_ATTR_D.REPL_WH_LEADTIME

Measure	Target Table	Target Column	Data Source
Replenishment Active Flag	W_PDS_REPL_ATTR_IT_LC_D	REPL_ACTIVE_FLAG	CASE WHEN W_INVENTORY_PRODUCT_ATTR_D.REPL_ACTIVE_FLAG IS NULL THEN CASE WHEN NVL(W_INVENTORY_PRODUCT_ATTR_D.REPL_DEACTIVATE_DT ,'2100-01-01') > current_dt THEN ''Y'' ELSE ''N'' END ELSE W_INVENTORY_PRODUCT_ATTR_D.REPL_ACTIVE_FLAG END
Exit Date	W_PDS_REPL_ATTR_IT_LC_D	EXIT_DATE	W_INVENTORY_PRODUCT_ATTR_D.EXIT_DATE
Promo Exclude Flg	W_PDS_REPL_ATTR_IT_LC_D	PROMO_EXCL_FLG	W_INVENTORY_PRODUCT_ATTR_D.PROMO_EXCL_FLG
Mkdn Exclude Flg	W_PDS_REPL_ATTR_IT_LC_D	MKDN_EXCL_FLG	W_INVENTORY_PRODUCT_ATTR_D.MKDN_EXCL_FLG
Replenishment Order Ctrl	W_PDS_REPL_ATTR_IT_LC_D	REPL_ORDER_CTRL	W_INVENTORY_PRODUCT_ATTR_D.REPL_ORDER_CTRL
Replenishment Method	W_PDS_REPL_ATTR_IT_LC_D	REPL_METHOD	W_INVENTORY_PRODUCT_ATTR_D.REPL_METHOD
Incr Percent	W_PDS_REPL_ATTR_IT_LC_D	INCR_PCT	W_INVENTORY_PRODUCT_ATTR_D.INCR_PCT
Min Supply Days	W_PDS_REPL_ATTR_IT_LC_D	MIN_SUPPLY_DAYS	W_INVENTORY_PRODUCT_ATTR_D.MIN_SUPPLY_DAYS
Max Supply Days	W_PDS_REPL_ATTR_IT_LC_D	MAX_SUPPLY_DAYS	W_INVENTORY_PRODUCT_ATTR_D.MAX_SUPPLY_DAYS
Time Supply Horizon	W_PDS_REPL_ATTR_IT_LC_D	TIME_SUPPLY_HORIZON	W_INVENTORY_PRODUCT_ATTR_D.TIME_SUPPLY_HORIZON
Inv Selling Days	W_PDS_REPL_ATTR_IT_LC_D	INV_SELLING_DAYS	W_INVENTORY_PRODUCT_ATTR_D.INV_SELLING_DAYS
Lost Sales Factor	W_PDS_REPL_ATTR_IT_LC_D	LOST_SALES_FACTOR	W_INVENTORY_PRODUCT_ATTR_D.LOST_SALES_FACTOR
Reject Store Order Flag	W_PDS_REPL_ATTR_IT_LC_D	REJECT_STORE_ORD_IND	W_INVENTORY_PRODUCT_ATTR_D.REJECT_STORE_ORD_IND
Non Scaling Flag	W_PDS_REPL_ATTR_IT_LC_D	NON_SCALING_IND	W_INVENTORY_PRODUCT_ATTR_D.NON_SCALING_IND
Max Scale Value	W_PDS_REPL_ATTR_IT_LC_D	MAX_SCALE_VALUE	W_INVENTORY_PRODUCT_ATTR_D.MAX_SCALE_VALUE
Terminal Stock Qty	W_PDS_REPL_ATTR_IT_LC_D	TERMINAL_STOCK_QTY	W_INVENTORY_PRODUCT_ATTR_D.TERMINAL_STOCK_QTY
Season Id	W_PDS_REPL_ATTR_IT_LC_D	SEASON_ID	W_INVENTORY_PRODUCT_ATTR_D.SEASON_ID
Phase Id	W_PDS_REPL_ATTR_IT_LC_D	PHASE_ID	W_INVENTORY_PRODUCT_ATTR_D.PHASE_ID
Last Review Date	W_PDS_REPL_ATTR_IT_LC_D	LAST_REVIEW_DATE	W_INVENTORY_PRODUCT_ATTR_D.LAST_REVIEW_DATE
Next Review Date	W_PDS_REPL_ATTR_IT_LC_D	NEXT_REVIEW_DATE	W_INVENTORY_PRODUCT_ATTR_D.NEXT_REVIEW_DATE

Measure	Target Table	Target Column	Data Source
Unit Tolerance	W_PDS_REPL_ATTR_IT_LC_D	UNIT_TOLERANCE	W_INVENTORY_PRODUCT_ATTR_D.UNIT_TOLERANCE
Percent Tolerance	W_PDS_REPL_ATTR_IT_LC_D	PCT_TOLERANCE	W_INVENTORY_PRODUCT_ATTR_D.PCT_TOLERANCE
Use Tolerance Flag	W_PDS_REPL_ATTR_IT_LC_D	USE_TOLERANCE_IND	W_INVENTORY_PRODUCT_ATTR_D.USE_TOLERANCE_IND
Last Delivery Date	W_PDS_REPL_ATTR_IT_LC_D	LAST_DELIVERY_DATE	W_INVENTORY_PRODUCT_ATTR_D.LAST_DELIVERY_DATE
Next Delivery Date	W_PDS_REPL_ATTR_IT_LC_D	NEXT_DELIVERY_DATE	W_INVENTORY_PRODUCT_ATTR_D.NEXT_DELIVERY_DATE
MBR Order Qty	W_PDS_REPL_ATTR_IT_LC_D	MBR_ORDER_QTY	W_INVENTORY_PRODUCT_ATTR_D.MBR_ORDER_QTY
Adj Pickup Lead Time	W_PDS_REPL_ATTR_IT_LC_D	ADJ_PICKUP_LEAD_TIME	W_INVENTORY_PRODUCT_ATTR_D.ADJ_PICKUP_LEAD_TIME
Adj Supp Lead Time	W_PDS_REPL_ATTR_IT_LC_D	ADJ_SUPP_LEAD_TIME	W_INVENTORY_PRODUCT_ATTR_D.ADJ_SUPP_LEAD_TIME
Tsf Po Link No	W_PDS_REPL_ATTR_IT_LC_D	TSF_PO_LINK_NO	W_INVENTORY_PRODUCT_ATTR_D.TSF_PO_LINK_NO
Last ROQ	W_PDS_REPL_ATTR_IT_LC_D	LAST_ROQ	W_INVENTORY_PRODUCT_ATTR_D.LAST_ROQ
Store Ord Multiple	W_PDS_REPL_ATTR_IT_LC_D	STORE_ORD_MULT	W_INVENTORY_PRODUCT_ATTR_D.STORE_ORD_MULT
Unit Cost	W_PDS_REPL_ATTR_IT_LC_D	UNIT_COST	W_INVENTORY_PRODUCT_ATTR_D.UNIT_COST
Supplier Lead Time	W_PDS_REPL_ATTR_IT_LC_D	SUPP_LEAD_TIME	W_INVENTORY_PRODUCT_ATTR_D.SUPP_LEAD_TIME
Inner Pack Size	W_PDS_REPL_ATTR_IT_LC_D	INNER_PACK_SIZE	W_INVENTORY_PRODUCT_ATTR_D.INNER_PACK_SIZE
Supplier Pack Size	W_PDS_REPL_ATTR_IT_LC_D	SUPP_PACK_SIZE	W_INVENTORY_PRODUCT_ATTR_D.SUPP_PACK_SIZE
Tier	W_PDS_REPL_ATTR_IT_LC_D	TIER	W_INVENTORY_PRODUCT_ATTR_D.TIER
Height	W_PDS_REPL_ATTR_IT_LC_D	HEIGHT	W_INVENTORY_PRODUCT_ATTR_D.HEIGHT
Round Lvl	W_PDS_REPL_ATTR_IT_LC_D	ROUND_LVL	W_INVENTORY_PRODUCT_ATTR_D.ROUND_LVL
Round To Inner Percent	W_PDS_REPL_ATTR_IT_LC_D	ROUND_TO_INNER_PCT	W_INVENTORY_PRODUCT_ATTR_D.ROUND_TO_INNER_PCT
Round To Case Percent	W_PDS_REPL_ATTR_IT_LC_D	ROUND_TO_CASE_PCT	W_INVENTORY_PRODUCT_ATTR_D.ROUND_TO_CASE_PCT
Round To Layer Percent	W_PDS_REPL_ATTR_IT_LC_D	ROUND_TO_LAYER_PCT	W_INVENTORY_PRODUCT_ATTR_D.ROUND_TO_LAYER_PCT
Round To Pallet Percent	W_PDS_REPL_ATTR_IT_LC_D	ROUND_TO_PALLET_PCT	W_INVENTORY_PRODUCT_ATTR_D.ROUND_TO_PALLET_PCT
Service Level Type	W_PDS_REPL_ATTR_IT_LC_D	SERVICE_LEVEL_TYPE	W_INVENTORY_PRODUCT_ATTR_D.SERVICE_LEVEL_TYPE
Tsf Zero SOH Flag	W_PDS_REPL_ATTR_IT_LC_D	TSF_ZERO_SOH_IND	W_INVENTORY_PRODUCT_ATTR_D.TSF_ZERO_SOH_IND

Measure	Target Table	Target Column	Data Source
Multiple Runs Per Day Flag	W_PDS_REPL_ATTR_IT_LC_D	MULT_RUNS_PER_DAY_IND	W_INVENTORY_PRODUCT_ATTR_D.MULT_RUNS_PER_DAY_IND
Add Lead Time Flag	W_PDS_REPL_ATTR_IT_LC_D	ADD_LEAD_TIME_IND	W_INVENTORY_PRODUCT_ATTR_D.ADD_LEAD_TIME_IND
Deposit Code	W_PDS_REPL_ATTR_IT_LC_D	DEPOSIT_CODE	W_INVENTORY_PRODUCT_ATTR_D.DEPOSIT_CODE
Proportional Tare Percent	W_PDS_REPL_ATTR_IT_LC_D	PROPORTIONAL_TARE_PCT	W_INVENTORY_PRODUCT_ATTR_D.PROPORTIONAL_TARE_PCT
Fixed Tare Value	W_PDS_REPL_ATTR_IT_LC_D	FIXED_TARE_VALUE	W_INVENTORY_PRODUCT_ATTR_D.FIXED_TARE_VALUE
Fixed Tare UOM	W_PDS_REPL_ATTR_IT_LC_D	FIXED_TARE_UOM	W_INVENTORY_PRODUCT_ATTR_D.FIXED_TARE_UOM
Return Policy	W_PDS_REPL_ATTR_IT_LC_D	RETURN_POLICY	W_INVENTORY_PRODUCT_ATTR_D.RETURN_POLICY
Stop Sale Flag	W_PDS_REPL_ATTR_IT_LC_D	STOP_SALE_IND	W_INVENTORY_PRODUCT_ATTR_D.STOP_SALE_IND
Report Code	W_PDS_REPL_ATTR_IT_LC_D	REPORT_CODE	W_INVENTORY_PRODUCT_ATTR_D.REPORT_CODE
Reference Date Type For Exit	W_PDS_REPL_ATTR_IT_LC_D	REFERENCE_DATE_TYPE_FOR_EXIT	W_INVENTORY_PRODUCT_ATTR_D.REFERENCE_DATE_TYPE_FOR_EXIT
Weeks To Exit	W_PDS_REPL_ATTR_IT_LC_D	WEEKS_TO_EXIT	W_INVENTORY_PRODUCT_ATTR_D.WEEKS_TO_EXIT
Optimize Flag	W_PDS_REPL_ATTR_IT_LC_D	OPTIMIZE_IND	W_INVENTORY_PRODUCT_ATTR_D.OPTIMIZE_IND

Supplier Mapping

The supplier data is loaded from the `PRODUCT.csv` file or from RMFCS. The product data load programs will insert the supplier information into the additional tables used below (as long as these tables are enabled during foundation loads).

Measure	Target Table	Target Column	Data Source
Supplier ID	W_PDS_SUPPLIER_D	SUPPLIER	W_PARTY_ORG_D .SUPPLIER_NUM
Supplier Desc	W_PDS_SUPPLIER_D	SUP_NAME	W_PARTY_ORG_D .ORG_NAME

Customer Segment Mapping

The customer segment data is loaded from the `W_RTL_CUSTSEG_DS.dat` file or from AI Foundation exports of the segment data back into the data warehouse table `W_RTL_CUSTSEG_D`. The data is exported to PDS for use in Assortment Planning, which has the ability to select a customer segment to use for AIF integrations like Demand Transference results.

Measure	Target Table	Target Column	Data Source
Segment ID	W_PDS_CUSTSEG_D	CUSTSEG_ID	W_RTL_CUSTSEG_D.CUSTSEG_ID

Measure	Target Table	Target Column	Data Source
Segment Name	W_PDS_CUSTSEG_D	CUSTSEG_NAME	NVL(W_RTL_CUSTSEG_D.CUSTSEG_NAME,'UNKNOWN SEGMENT')
Segment Desc	W_PDS_CUSTSEG_D	CUSTSEG_DESC	W_RTL_CUSTSEG_D.CUSTSEG_DESC
Segment Type	W_PDS_CUSTSEG_D	CUSTSEG_TYPE	W_RTL_CUSTSEG_D.CUSTSEG_TYPE
Segment Source Type	W_PDS_CUSTSEG_D	CUSTSEG_SRC_TYPE	W_RTL_CUSTSEG_D.CUSTSEG_SRC_TYPE
Segment Organization Hierarchy Key	W_PDS_CUSTSEG_D	ORG_HIER_KEY	W_RTL_CUSTSEG_D.ORG_HIER_KEY

Custom Flex Attribute Mapping

Custom flex attributes (also known as CFAS) data comes from RMFCS. This data is passed through the data warehouse and transformed to match RPAS data requirements. The CFAS data is exported to the planning integration schema in 9 tables: 3 sets of tables for each intersection of item, location, and item/location, with each set broken out by datatype (string, number, date). Each set of tables will uses anidentical structure where the source data in the data warehouse (which is in multiple columns) is pivoted into a single column having all the attribute values for a given intersection and datatype.

Specifically for location attributes, the same table may optionally include location attributes from other sources which can then be used to extend your Planning applications with that data. CFAS attributes are exported automatically from the data warehouse when they are integrated from MFCS; but for the other types, it requires additional configurations. The reason for requiring configurations is that location attributes data does not come with associated group IDs and group names that Planning requires to use the data, so the implementer must separately generate these groupings first.

The list of supported tables on this integration are below:

Source Tables	Target Table
W_INT_ORG_D_CFA W_INT_ORG_ATTR_D W_ORGANIZATION_FLEX_D	W_PDS_ORG_ATTR_STR_D
W_INT_ORG_D_CFA	W_PDS_ORG_ATTR_NBR_D
W_INT_ORG_D_CFA	W_PDS_ORG_ATTR_DT_D
W_PRODUCT_D_CFA	W_PDS_PRODUCT_ATTR_STR_D
W_PRODUCT_D_CFA	W_PDS_PRODUCT_ATTR_NBR_D
W_PRODUCT_D_CFA	W_PDS_PRODUCT_ATTR_DT_D
W_RTL_IT_LC_D_CFA	W_PDS_PROD_ORG_ATTR_STR_D
W_RTL_IT_LC_D_CFA	W_PDS_PROD_ORG_ATTR_NBR_D
W_RTL_IT_LC_D_CFA	W_PDS_PROD_ORG_ATTR_DT_D

All of the tables include an `ATTR_ID` column that is automatically generated from either the CFAS attribute group name or the user-supplied group names, stripped of unsupported characters for ID columns. They also have an `ATTR_VALUE` column containing the attribute values from the source table, which is a combination of all the values in all columns having the same data type (CHAR, NUM, DATE). ATTR_VALUE for string types are a trimmed version with unsupported characters removed. Also specific to string types, there is an ATTR_VALUE_DESC column which has the unmodified string value from the source table.

To leverage location string attributes for non-CFAS data, you need to load the location group definitions to the common translation lookup table in the data warehouse, W_DOMAIN_MEMBER_LKP_TL. From APEX, you may insert the group names to the staging table W_DOMAIN_MEMBER_DS_TL and then run the associated POM job W_DOMAIN_MEMBER_LKP_TL_JOB to populate the target table. You need to populate the columns with specific values as described below:

Column	Usage
DOMAIN_CODE	Use RTL_ORG_ATTR for columns in W_INT_ORG_ATTR_D and RTL_ORG_FLEX for columns in W_ORGANIZATION_FLEX_D
DOMAIN_MEMBER_CODE	Specify the exact column in the source table having this attribute in it, such as FLEX1_CHAR_VALUE
DOMAIN_MEMBER_NAME	Specify the name of the attribute group, such as Climate
LANGUAGE_CODE	Specify the primary language code used for all translated lookup data, such as US
SRC_LANGUAGE_CODE	Specify the primary language code used for all translated lookup data, such as US

Sales Mapping

Data for sales is loaded from the SALES.csv file or from RMFCS (Sales Audit). The primary data warehouse table is the week-level aggregate generated by the historical and daily load processes. All data mappings in this area are split out by retail type. Any measure having reg/pro/clr in the name are being filtered on that retail type code as part of the export. When you provide input data to RAP, you specify the retail type code as R, P, or C, and those values are used here to determine the output. A custom 4th option (using type code O for Other) is also allowed, as long as you extend the W_XACT_TYPE_D dimension in the data warehouse to have the extra type code. Other sales are only included in the Total Sales measures in the PDS export. The data only includes non-pack item sales, as it expects pack sales to be spread to their component level when used.

Measure	Target Table	Target Column	Data Source
Gross Reg Sales Units	W_PDS_SLS_IT_LC_WK_A	SALES_REG_UNITS	W_RTL_SLS_IT_LC_WK_A.SLS_QTY + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY
Gross Pro Sales Units	W_PDS_SLS_IT_LC_WK_A	SALES_PRO_UNITS	W_RTL_SLS_IT_LC_WK_A.SLS_QTY + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY
Gross Clr Sales Units	W_PDS_SLS_IT_LC_WK_A	SALES_CLR_UNITS	W_RTL_SLS_IT_LC_WK_A.SLS_QTY + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY

Measure	Target Table	Target Column	Data Source
Gross Reg Sales Cost	W_PDS_SLS_IT_LC_WK_A	SALES_REG_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-SLS_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-SLSPK_PROF_AMT)
Gross Pro Sales Cost	W_PDS_SLS_IT_LC_WK_A	SALES_PRO_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-SLS_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-SLSPK_PROF_AMT)
Gross Clr Sales Cost	W_PDS_SLS_IT_LC_WK_A	SALES_CLR_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-SLS_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-SLSPK_PROF_AMT)
Gross Reg Sales Retail	W_PDS_SLS_IT_LC_WK_A	SALES_REG_RETAIL	W_RTL_SLS_IT_LC_WK_A.SLS_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT
Gross Pro Sales Retail	W_PDS_SLS_IT_LC_WK_A	SALES_PRO_RETAIL	W_RTL_SLS_IT_LC_WK_A.SLS_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT
Gross Clr Sales Retail	W_PDS_SLS_IT_LC_WK_A	SALES_CLR_RETAIL	W_RTL_SLS_IT_LC_WK_A.SLS_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT
Gross Reg Sales Tax	W_PDS_SLS_IT_LC_WK_A	SALES_REG_TAX	W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT
Gross Pro Sales Tax	W_PDS_SLS_IT_LC_WK_A	SALES_PRO_TAX	W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT
Gross Clr Sales Tax	W_PDS_SLS_IT_LC_WK_A	SALES_CLR_TAX	W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT
Net Reg Sales Units	W_PDS_SLS_IT_LC_WK_A	NET_SALES_REG_UNITS	(W_RTL_SLS_IT_LC_WK_A.SLS_QTY-RET_QTY) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY-RETPK_QTY)
Net Pro Sales Units	W_PDS_SLS_IT_LC_WK_A	NET_SALES_PRO_UNITS	(W_RTL_SLS_IT_LC_WK_A.SLS_QTY-RET_QTY) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY-RETPK_QTY)
Net Clr Sales Units	W_PDS_SLS_IT_LC_WK_A	NET_SALES_CLR_UNITS	(W_RTL_SLS_IT_LC_WK_A.SLS_QTY-RET_QTY) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY-RETPK_QTY)

Measure	Target Table	Target Column	Data Source
Net Reg Sales Cost	W_PDS_SLS_IT_LC_WK_A	NET_SALES_REG_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_PROFIT_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_PROF_AMT-RETPK_PROF_AMT)
Net Pro Sales Cost	W_PDS_SLS_IT_LC_WK_A	NET_SALES_PRO_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_PROFIT_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_PROF_AMT-RETPK_PROF_AMT)
Net Clr Sales Cost	W_PDS_SLS_IT_LC_WK_A	NET_SALES_CLR_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_PROFIT_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_PROF_AMT-RETPK_PROF_AMT)
Net Reg Sales Retail	W_PDS_SLS_IT_LC_WK_A	NET_SALES_REG_RETAIL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT)
Net Pro Sales Retail	W_PDS_SLS_IT_LC_WK_A	NET_SALES_PRO_RETAIL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT)
Net Clr Sales Retail	W_PDS_SLS_IT_LC_WK_A	NET_SALES_CLR_RETAIL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT)
Net Reg Sales Retail Excluding VAT	W_PDS_SLS_IT_LC_WK_A	NET_SALES_REG_RETAIL_VAT_EXCL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)
Net Pro Sales Retail Excluding VAT	W_PDS_SLS_IT_LC_WK_A	NET_SALES_PRO_RETAIL_VAT_EXCL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)

Measure	Target Table	Target Column	Data Source
Net Clr Sales Retail Excluding VAT	W_PDS_SLS_IT_LC_WK_A	NET_SALES_CLR_RETAIL_VAT_EXCL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)
Net Reg Sales Tax	W_PDS_SLS_IT_LC_WK_A	NET_SALES_REG_TAX	(W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)
Net Pro Sales Tax	W_PDS_SLS_IT_LC_WK_A	NET_SALES_PRO_TAX	(W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)
Net Clr Sales Tax	W_PDS_SLS_IT_LC_WK_A	NET_SALES_CLR_TAX	(W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)
Returns Reg Units	W_PDS_SLS_IT_LC_WK_A	RETURNS_REG_UNITS	W_RTL_SLS_IT_LC_WK_A.RET_QTY + W_RTL_SLSPK_IT_LC_WK_A.RETPK_QTY
Returns Pro Units	W_PDS_SLS_IT_LC_WK_A	RETURNS_PRO_UNITS	W_RTL_SLS_IT_LC_WK_A.RET_QTY + W_RTL_SLSPK_IT_LC_WK_A.RETPK_QTY
Returns Clr Units	W_PDS_SLS_IT_LC_WK_A	RETURNS_CLR_UNITS	W_RTL_SLS_IT_LC_WK_A.RET_QTY + W_RTL_SLSPK_IT_LC_WK_A.RETPK_QTY
Returns Reg Cost	W_PDS_SLS_IT_LC_WK_A	RETURNS_REG_COST	(W_RTL_SLS_IT_LC_WK_A.RET_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.RETPK_AMT-RETPK_PROF_AMT)
Returns Pro Cost	W_PDS_SLS_IT_LC_WK_A	RETURNS_PRO_COST	(W_RTL_SLS_IT_LC_WK_A.RET_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.RETPK_AMT-RETPK_PROF_AMT)
Returns Clr Cost	W_PDS_SLS_IT_LC_WK_A	RETURNS_CLR_COST	(W_RTL_SLS_IT_LC_WK_A.RET_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.RETPK_AMT-RETPK_PROF_AMT)
Returns Reg Retail	W_PDS_SLS_IT_LC_WK_A	RETURNS_REG_RETAIL	W_RTL_SLS_IT_LC_WK_A.RET_AMT + W_RTL_SLSPK_IT_LC_WK_A.RETPK_AMT
Returns Pro Retail	W_PDS_SLS_IT_LC_WK_A	RETURNS_PRO_RETAIL	W_RTL_SLS_IT_LC_WK_A.RET_AMT + W_RTL_SLSPK_IT_LC_WK_A.RETPK_AMT

Measure	Target Table	Target Column	Data Source
Returns Clr Retail	W_PDS_SLS_IT_LC_WK_A	RETURNS_CLR_RETAIL	W_RTL_SLS_IT_LC_WK_A.RET_AMT + W_RTL_SLSPK_IT_LC_WK_A.RETPK_AMT
Returns Reg Tax	W_PDS_SLS_IT_LC_WK_A	RETURNS_REG_TAX	W_RTL_SLS_IT_LC_WK_A.RET_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.RETPK_TAX_AMT
Returns Pro Tax	W_PDS_SLS_IT_LC_WK_A	RETURNS_PRO_TAX	W_RTL_SLS_IT_LC_WK_A.RET_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.RETPK_TAX_AMT
Returns Clr Tax	W_PDS_SLS_IT_LC_WK_A	RETURNS_CLR_TAX	W_RTL_SLS_IT_LC_WK_A.RET_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.RETPK_TAX_AMT
Total Gross Sales Units	W_PDS_SLS_IT_LC_WK_A	SALES_TOTAL_UNITS	W_RTL_SLS_IT_LC_WK_A.SLS_QTY + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY
Total Gross Sales Cost	W_PDS_SLS_IT_LC_WK_A	SALES_TOTAL_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-SLS_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-SLSPK_PROF_AMT)
Total Gross Sales Retail	W_PDS_SLS_IT_LC_WK_A	SALES_TOTAL_RETAIL	W_RTL_SLS_IT_LC_WK_A.SLS_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT
Total Gross Sales Tax	W_PDS_SLS_IT_LC_WK_A	SALES_TOTAL_TAX	W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT
Total Net Sales Units	W_PDS_SLS_IT_LC_WK_A	NET_SALES_TOTAL_UNITS	(W_RTL_SLS_IT_LC_WK_A.SLS_QTY-RET_QTY) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY-RETPK_QTY)
Total Net Sales Cost	W_PDS_SLS_IT_LC_WK_A	NET_SALES_TOTAL_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) - (W_RTL_SLS_IT_LC_WK_A.SLS_PROFIT_AMT-RET_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT) - (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_PROF_AMT-RETPK_PROF_AMT)
Total Net Sales Retail	W_PDS_SLS_IT_LC_WK_A	NET_SALES_TOTAL_RETAIL	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-RET_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-RETPK_AMT)
Total Net Sales Tax	W_PDS_SLS_IT_LC_WK_A	NET_SALES_TOTAL_TAX	(W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT-RET_TAX_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT-RETPK_TAX_AMT)

Measure	Target Table	Target Column	Data Source
Custom Measures 1 - 20	W_PDS_SLS_IT_LC_WK_A	FLEX1_NUM_VALUE through FLEX20_NUM_VALUE	W_RTL_SLS_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_SLS_IT_LC_WK_A.FLEX20_NUM_VALUE

Gross Sales Mapping

A separate sales aggregate is also provided for Inventory Planning Optimization Cloud Service-Demand Forecasting (IPOCS-Demand Forecasting), which filters and aggregates the sales differently from the base extract for the other Planning applications. This export includes only gross sales and has a single set of measure columns with a separate field for the retail type (reg/pro/clr). The data is filtered to include only sales for non-warehouse locations. The data only includes non-pack item sales, as it expects pack sales to be spread to their component level when used.

Measure	Target Table	Target Column	Data Source
Gross Sales Units	W_PDS_GRS_SLS_IT_LC_WK_A	SALES_UNITS	W_RTL_SLS_IT_LC_WK_A.SLS_QTY + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_QTY
Gross Sales Cost	W_PDS_GRS_SLS_IT_LC_WK_A	SALES_COST	(W_RTL_SLS_IT_LC_WK_A.SLS_AMT-SLS_PROFIT_AMT) + (W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT-SLSPK_PROF_AMT)
Gross Sales Retail	W_PDS_GRS_SLS_IT_LC_WK_A	SALES_RETAIL	W_RTL_SLS_IT_LC_WK_A.SLS_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_AMT
Gross Sales Tax	W_PDS_GRS_SLS_IT_LC_WK_A	SALES_TAX	W_RTL_SLS_IT_LC_WK_A.SLS_TAX_AMT + W_RTL_SLSPK_IT_LC_WK_A.SLSPK_TAX_AMT

Inventory Position Mapping

Data is loaded from the `INVENTORY.csv` and `INVENTORY_PACK.csv` files, or from RMFCS. The primary data warehouse table is the week-level aggregates generated by the historical load process. Only data for stockholding locations (`STOCKHOLDING_FLG=Y`) is exported to PDS. Pack inventory rows are combined with non-pack inventory and use the `PACK_FLG` column to differentiate the records, so each set of data sources provided is for the non-pack and pack inventory rows, respectively.

Measure	Target Table	Target Column	Data Source
Inventory Units	W_PDS_INV_IT_LC_WK_A	REGULAR_INVENTORY_UNITS	W_RTL_INV_IT_LC_WK_A.INV_SOH_QTY + INV_IN_TRAN_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_SOH_QTY + INVPK_IN_TRAN_QTY

Measure	Target Table	Target Column	Data Source
Inventory Cost	W_PDS_INV_IT_LC_WK_A	REGULAR_INVENTORY_COST	W_RTL_INV_IT_LC_WK_A.INV_SOH_COST_AMT + INV_IN_TRAN_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_SOH_COST_AMT + INVPK_IN_TRAN_COST_AMT
Inventory Retail	W_PDS_INV_IT_LC_WK_A	REGULAR_INVENTORY_RETAIL	W_RTL_INV_IT_LC_WK_A.INV_SOH_RTL_AMT + INV_IN_TRAN_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_SOH_RTL_AMT + INVPK_IN_TRAN_RTL_AMT
Unit Cost	W_PDS_INV_IT_LC_WK_A	UNIT_COST	W_RTL_INV_IT_LC_WK_A.INV_UNIT_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_UNIT_COST_AMT
Average Cost	W_PDS_INV_IT_LC_WK_A	AV_COST	W_RTL_INV_IT_LC_WK_A.INV_AVG_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_AVG_COST_AMT
Unit Retail	W_PDS_INV_IT_LC_WK_A	UNIT_RETAIL	W_RTL_INV_IT_LC_WK_A.INV_UNIT_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_UNIT_RTL_AMT
Custom Measures 1 - 20	W_PDS_INV_IT_LC_WK_A	FLEX1_NUM_VALUE through FLEX20_NUM_VALUE	W_RTL_INV_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_INV_IT_LC_WK_A.FLEX20_NUM_VALUE W_RTL_INVPK_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_INVPK_IT_LC_WK_A.FLEX20_NUM_VALUE
On Hand Units	W_PDS_INV_IT_LC_WK_A	INV_SOH_QTY	W_RTL_INV_IT_LC_WK_A.INV_SOH_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_SOH_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_SOH_QTY
Customer Order Reserved Units	W_PDS_INV_IT_LC_WK_A	INV_CO_RSV_QTY	W_RTL_INV_IT_LC_WK_A. INV_CO_RSV_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_CO_RSV_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_CO_RSV_QTY
Backorder Units	W_PDS_INV_IT_LC_WK_A	INV_CO_BO_RSV_QTY	W_RTL_INV_IT_LC_WK_A. INV_CO_BO_RSV_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_CO_BO_RSV_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_CO_BO_RSV_QTY

Measure	Target Table	Target Column	Data Source
Finisher Units	W_PDS_INV_IT_LC_WK_A	FINISHER_QTY	W_RTL_INV_IT_LC_WK_A.FINISHER_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_FINISHER_QTY
In Progress Sales	W_PDS_INV_IT_LC_WK_A	INV_IP_SLS_QTY	W_RTL_INV_IT_LC_WK_A.INV_IP_SLS_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_IP_SLS_QTY
On Order Units	W_PDS_INV_IT_LC_WK_A	INV_ON_ORD_QTY	W_RTL_INV_IT_LC_WK_A.INV_ON_ORD_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_ON_ORD_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_ON_ORD_QTY
On Order Cost	W_PDS_INV_IT_LC_WK_A	INV_ON_ORD_COST_AMT	W_RTL_INV_IT_LC_WK_A.INV_ON_ORD_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_ON_ORD_COST_AMT
On Order Retail	W_PDS_INV_IT_LC_WK_A	INV_ON_ORD_RTL_AMT	W_RTL_INV_IT_LC_WK_A.INV_ON_ORD_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_ON_ORD_RTL_AMT
In Transit Units	W_PDS_INV_IT_LC_WK_A	INV_IN_TRAN_QTY	W_RTL_INV_IT_LC_WK_A.INV_IN_TRAN_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_IN_TRAN_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_IN_TRAN_QTY
In Transit Cost	W_PDS_INV_IT_LC_WK_A	INV_IN_TRAN_COST_AMT	W_RTL_INV_IT_LC_WK_A.INV_IN_TRAN_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_ IN_TRAN_COST_AMT
In Transit Retail	W_PDS_INV_IT_LC_WK_A	INV_IN_TRAN_RTL_AMT	W_RTL_INV_IT_LC_WK_A.INV_IN_TRAN_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_ IN_TRAN_RTL_AMT
Transfer Reserved Units	W_PDS_INV_IT_LC_WK_A	INV_TSF_RESV_QTY	W_RTL_INV_IT_LC_WK_A.INV_TSF_RESV_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_TSF_RESV_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_TSF_RESV_QTY
Transfer Reserved Cost	W_PDS_INV_IT_LC_WK_A	INV_TSF_RESV_COST_AMT	W_RTL_INV_IT_LC_WK_A.INV_TSF_RESV_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_TSF_RESV_COST_AMT
Transfer Reserved Retail	W_PDS_INV_IT_LC_WK_A	INV_TSF_RESV_RTL_AMT	W_RTL_INV_IT_LC_WK_A.INV_TSF_RESV_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_TSF_RESV_RTL_AMT

Measure	Target Table	Target Column	Data Source
Transfer Expected Units	W_PDS_INV_IT_LC_WK_A	INV_TSF_EXP_QTY	W_RTL_INV_IT_LC_WK_A.INV_TSF_EXP_QTY - W_RTL_INV_IT_LC_WK_A.INV_COMP_TSF_EXP_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_TSF_EXP_QTY
Transfer Expected Cost	W_PDS_INV_IT_LC_WK_A	INV_TSF_EXP_COST_AMT	W_RTL_INV_IT_LC_WK_A.INV_TSF_EXP_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_TSF_EXP_COST_AMT
Transfer Expected Retail	W_PDS_INV_IT_LC_WK_A	INV_TSF_EXP_RTL_AMT	W_RTL_INV_IT_LC_WK_A.INV_TSF_EXP_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_TSF_EXP_RTL_AMT
RTV Reserved Units	W_PDS_INV_IT_LC_WK_A	INV_RTV_RESV_QTY	W_RTL_INV_IT_LC_WK_A.INV_RTV_RESV_QTY W_RTL_INVPK_IT_LC_WK_A.INVPK_RTV_RESV_QTY
RTV Reserved Cost	W_PDS_INV_IT_LC_WK_A	INV_RTV_RESV_COST_AMT	W_RTL_INV_IT_LC_WK_A.INV_RTV_RESV_COST_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_RTV_RESV_COST_AMT
RTV Reserved Retail	W_PDS_INV_IT_LC_WK_A	INV_RTV_RESV_RTL_AMT	W_RTL_INV_IT_LC_WK_A.INV_RTV_RESV_RTL_AMT W_RTL_INVPK_IT_LC_WK_A.INVPK_RTV_RESV_RTL_AMT
Pack Flag	W_PDS_INV_IT_LC_WK_A	PACK_FLG	Fixed value of `N` or `Y` depending on the source of the inventory record (`INV` or `INVPK` table)

On Order Mapping

Data is loaded from the `ORDER_HEAD.csv` and `ORDER_DETAIL.csv` files or from RMFCS. Purchase order data is transformed from the raw order line details into a forward-looking total on-order amount based on the OTB end-of-week date on the order. The calendar date on the export is further altered based on the parameter `PDS_EXPORT_DAILY_ONORD` in `C_ODI_PARAM_VW` to either allow or prevent non-week-ending dates. Data is also filtered to remove orders not flagged as Include On Order.

Measure	Target Table	Target Column	Data Source
On Order Units	W_PDS_PO_ONORD_IT_LC_WK_A	ON_ORDER_UNITS	W_RTL_PO_ONORD_IT_LC_DY_F.PO_ONORD_QTY
On Order Cost	W_PDS_PO_ONORD_IT_LC_WK_A	ON_ORDER_COST	W_RTL_PO_ONORD_IT_LC_DY_F.PO_ONORD_COST_AMT_LCL
On Order Retail	W_PDS_PO_ONORD_IT_LC_WK_A	ON_ORDER_RETAIL	W_RTL_PO_ONORD_IT_LC_DY_F.PO_ONORD_RTL_AMT_LCL

Measure	Target Table	Target Column	Data Source
Custom Measures 1 - 20	W_PDS_PO_ONORD_IT_LC_WK_A	FLEX1_NUM_VALUE through FLEX20_NUM_VALUE	W_RTL_PO_ONORD_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_PO_ONORD_IT_LC_WK_A.FLEX20_NUM_VALUE

Markdown Mapping

Data is loaded from the `MARKDOWN.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Regular Markdown	W_PDS_MKDN_IT_LC_WK_A	REG_MARKDOWN_RETAIL	W_RTL_MKDN_IT_LC_WK_A.MKDN_AMT where retail type = R
Promotion Markdown (Regular)	W_PDS_MKDN_IT_LC_WK_A	PROMO_MARKDOWN_RETAIL_REG	W_RTL_MKDN_IT_LC_WK_A.MKDN_AMT where retail type = P and CLEARANCE_FLG = N
Promotion Markdown (Clearance)	W_PDS_MKDN_IT_LC_WK_A	PROMO_MARKDOWN_RETAIL_CLEAR	W_RTL_MKDN_IT_LC_WK_A.MKDN_AMT where retail type = P and CLEARANCE_FLG = Y
Clearance Markdown	W_PDS_MKDN_IT_LC_WK_A	CLEAR_MARKDOWN_RETAIL	W_RTL_MKDN_IT_LC_WK_A.MKDN_AMT where retail type = C
Markup	W_PDS_MKDN_IT_LC_WK_A	MARKUP	W_RTL_MKDN_IT_LC_WK_A.MKUP_AMT where retail type = R
Markup Cancel	W_PDS_MKDN_IT_LC_WK_A	MARKUP_CANCEL	W_RTL_MKDN_IT_LC_WK_A.MKUP_CAN_AMT
Markdown Cancel	W_PDS_MKDN_IT_LC_WK_A	MARKDOWN_CANCEL	W_RTL_MKDN_IT_LC_WK_A.MKDN_CAN_AMT
Intercompany Markup	W_PDS_MKDN_IT_LC_WK_A	INTERCOMPANY_MARKUP	W_RTL_MKDN_IT_LC_WK_A.MKUP_AMT where retail type = I
Intercompany Markdown	W_PDS_MKDN_IT_LC_WK_A	INTERCOMPANY_MARKDOWN	W_RTL_MKDN_IT_LC_WK_A.MKDN_AMT where retail type = I
Custom Measures 1 - 20	W_PDS_MKDN_IT_LC_WK_A	FLEX1_NUM_VALUE through FLEX20_NUM_VALUE	W_RTL_MKDN_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_MKDN_IT_LC_WK_A.FLEX20_NUM_VALUE

Wholesale/Franchise Mapping

Data is loaded from the `SALES_WF.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Franchise Sales Units	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_SALES_UNITS	W_RTL_SLSWF_IT_LC_WK_A.SLSWF_QTY

Measure	Target Table	Target Column	Data Source
Franchise Sales Cost	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_SALES_COST	W_RTL_SLSWF_IT_LC_WK_A.SLSWF_ACQ_COST_AMT
Franchise Sales Retail	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_SALES_RETAIL	W_RTL_SLSWF_IT_LC_WK_A.SLSWF_AMT
Franchise Sales Tax	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_SALES_TAX	W_RTL_SLSWF_IT_LC_WK_A.SLSWF_TAX_AMT
Franchise Returns Units	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_RETURNS_UNITS	W_RTL_SLSWF_IT_LC_WK_A.RETWF_QTY
Franchise Returns Cost	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_RETURNS_COST	W_RTL_SLSWF_IT_LC_WK_A.RETWF_ACQ_COST_AMT
Franchise Returns Retail	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_RETURNS_RETAIL	W_RTL_SLSWF_IT_LC_WK_A.RETWF_AMT
Franchise Returns Tax	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_RETURNS_TAX	W_RTL_SLSWF_IT_LC_WK_A.RETWF_TAX_AMT
Franchise Restocking Fee	W_PDS_SLSWF_IT_LC_WK_A	FRANCHISE_RESTOCK_FEE	W_RTL_SLSWF_IT_LC_WK_A.RETWF_RSTK_FEE_AMT
Franchise Markdown	W_PDS_SLSWF_IT_LC_WK_A	WF_MARKDOWN_RETAIL	W_RTL_SLSWF_IT_LC_WK_A.SLSWF_MKDN_AMT - RETWF_MKDN_AMT
Franchise Markup	W_PDS_SLSWF_IT_LC_WK_A	WF_MARKUP_RETAIL	W_RTL_SLSWF_IT_LC_WK_A.SLSWF_MKUP_AMT - RETWF_MKUP_AMT

Inventory Adjustments Mapping

Data is loaded from the `ADJUSTMENT.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Shrink Units	W_PDS_INVADJ_IT_LC_WK_A	SHRINK_UNITS	W_RTL_INVADJ_IT_LC_WK_A.INVADJ_QTY where adj type = 22
Shrink Cost	W_PDS_INVADJ_IT_LC_WK_A	SHRINK_COST	W_RTL_INVADJ_IT_LC_WK_A.INVADJ_COST_AMT where adj type = 22
Shrink Retail	W_PDS_INVADJ_IT_LC_WK_A	SHRINK_RETAIL	W_RTL_INVADJ_IT_LC_WK_A.INVADJ_RTL_AMT where adj type = 22
Non-Shrink Adjustments Units	W_PDS_INVADJ_IT_LC_WK_A	NON_SHRINK_ADJ_UNITS	W_RTL_INVADJ_IT_LC_WK_A.INVADJ_QTY where adj type = 23
Non-Shrink Adjustments Cost	W_PDS_INVADJ_IT_LC_WK_A	NON_SHRINK_ADJ_COST	W_RTL_INVADJ_IT_LC_WK_A.INVADJ_COST_AMT where adj type = 23
Non-Shrink Adjustments Retail	W_PDS_INVADJ_IT_LC_WK_A	NON_SHRINK_ADJ_RETAIL	W_RTL_INVADJ_IT_LC_WK_A.INVADJ_RTL_AMT where adj type = 23

Inventory Receipts Mapping

Data is loaded from the `RECEIPT.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
PO Receipt Units	W_PDS_INVRC_IT_LC_WK_A	PO_RECEIPT_UNITS	W_RTL_INVRC_IT_LC_WK_A.INVRC_QTY where rcpt type code = 20
PO Receipt Cost	W_PDS_INVRC_IT_LC_WK_A	PO_RECEIPT_COST	W_RTL_INVRC_IT_LC_WK_A.INVRC_COST_AMT where rcpt type code = 20
PO Receipt Retail	W_PDS_INVRC_IT_LC_WK_A	PO_RECEIPT_RETAIL	W_RTL_INVRC_IT_LC_WK_A.INVRC_RTL_AMT where rcpt type code = 20
Transfer/ Allocation Receipt Units	W_PDS_INVRC_IT_LC_WK_A	TSF_RECEIPT_UNITS	W_RTL_INVRC_IT_LC_WK_A.INVRC_QTY where rcpt type code = 44~A or 44~T
Transfer/ Allocation Receipt Cost	W_PDS_INVRC_IT_LC_WK_A	TSF_RECEIPT_COST	W_RTL_INVRC_IT_LC_WK_A.INVRC_COST_AMT where rcpt type code = 44~A or 44~T
Transfer/ Allocation Receipt Retail	W_PDS_INVRC_IT_LC_WK_A	TSF_RECEIPT_RETAIL	W_RTL_INVRC_IT_LC_WK_A.INVRC_RTL_AMT where rcpt type code = 44~A or 44~T
PO Receipt Custom Measures 1 - 20	W_PDS_INVRC_IT_LC_WK_A	PO_FLEX1_NUM_VALUE through PO_FLEX20_NUM_VALUE	W_RTL_INVRC_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_INVRC_IT_LC_WK_A.FLEX20_NUM_VALUE where rcpt type code = 20
Transfer Receipt Custom Measures 1 - 20	W_PDS_INVRC_IT_LC_WK_A	TSF_FLEX1_NUM_VALUE through TSF_FLEX20_NUM_VALUE	W_RTL_INVRC_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_INVRC_IT_LC_WK_A.FLEX20_NUM_VALUE where rcpt type code = 44~A or 44~T

Inventory Transfers Mapping

Data is loaded from the `TRANSFER.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Transfer Type	W_PDS_INVTSF_IT_LC_WK_A	TSF_TYPE	W_XACT_TYPE_D.W_XACT_TYPE_CODE in (N,B,I) (for normal/book/intercompany tsfs)
Transfer Inbound Units	W_PDS_INVTSF_IT_LC_WK_A	TSF_IN_UNITS	W_RTL_INVTSF_IT_LC_WK_A.TSF_TO_LOC_QTY
Transfer Inbound Cost	W_PDS_INVTSF_IT_LC_WK_A	TSF_IN_COST	W_RTL_INVTSF_IT_LC_WK_A.TSF_TO_LOC_COST_AMT
Transfer Inbound Retail	W_PDS_INVTSF_IT_LC_WK_A	TSF_IN_RETAIL	W_RTL_INVTSF_IT_LC_WK_A.TSF_TO_LOC_RTL_AMT
Transfer Outbound Units	W_PDS_INVTSF_IT_LC_WK_A	TSF_OUT_UNITS	W_RTL_INVTSF_IT_LC_WK_A.TSF_FROM_LOC_QTY
Transfer Outbound Cost	W_PDS_INVTSF_IT_LC_WK_A	TSF_OUT_COST	W_RTL_INVTSF_IT_LC_WK_A.TSF_FROM_LOC_COST_AMT

Measure	Target Table	Target Column	Data Source
Transfer Outbound Retail	W_PDS_INVTSF_IT_LC_WK_A	TSF_OUT_RETAIL	W_RTL_INVTSF_IT_LC_WK_A.TSF_FROM_LOC_RTL_AMT

Inventory RTVs Mapping

Data is loaded from the `RTV.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
RTV Units	W_PDS_INVRTV_IT_LC_WK_A	RTV_UNITS	W_RTL_INVRTV_IT_LC_WK_A.RTV_QTY
RTV Cost	W_PDS_INVRTV_IT_LC_WK_A	RTV_COST	W_RTL_INVRTV_IT_LC_WK_A.RTV_COST_AMT
RTV Retail	W_PDS_INVRTV_IT_LC_WK_A	RTV_RETAIL	W_RTL_INVRTV_IT_LC_WK_A.RTV_RTL_AMT
RTV Restocking Fee	W_PDS_INVRTV_IT_LC_WK_A	RTV_RESTOCK_FEE	W_RTL_INVRTV_IT_LC_WK_A.RTV_RSTCK_COST_AMT

Inventory Reclass Mapping

Data is loaded from the `INV_RECLASS.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Reclass In Units	W_PDS_INVRECLASS_IT_LC_WK_A	RECLASS_IN_UNITS	W_RTL_INVRCL_IT_LC_WK_A.RCL_TO_LOC_QTY
Reclass In Cost	W_PDS_INVRECLASS_IT_LC_WK_A	RECLASS_IN_COST	W_RTL_INVRCL_IT_LC_WK_A.RCL_TO_LOC_COST_AMT
Reclass In Retail	W_PDS_INVRECLASS_IT_LC_WK_A	RECLASS_IN_RETAIL	W_RTL_INVRCL_IT_LC_WK_A.RCL_TO_LOC_RTL_AMT
Reclass Out Units	W_PDS_INVRECLASS_IT_LC_WK_A	RECLASS_OUT_UNITS	W_RTL_INVRCL_IT_LC_WK_A.RCL_FROM_LOC_QTY
Reclass Out Cost	W_PDS_INVRECLASS_IT_LC_WK_A	RECLASS_OUT_COST	W_RTL_INVRCL_IT_LC_WK_A.RCL_FROM_LOC_COST_AMT
Reclass Out Retail	W_PDS_INVRECLASS_IT_LC_WK_A	RECLASS_OUT_RETAIL	W_RTL_INVRCL_IT_LC_WK_A.RCL_FROM_LOC_RTL_AMT

Deal Income Mapping

Data is loaded from the `DEAL_INCOME.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Deal Income Sales Based	W_PDS_DEALINC_IT_LC_WK_A	DEAL_INCOME_SALES	W_RTL_DEALINC_IT_LC_WK_A.DEAL_SLS_COST_AMT
Deal Income Purchases Based	W_PDS_DEALINC_IT_LC_WK_A	DEAL_INCOME_PURCHASES	W_RTL_DEALINC_IT_LC_WK_A.DEAL_PRCH_COST_AMT
Custom Measures 1 - 20	W_PDS_ DEALINC_IT_LC_WK_A	FLEX1_NUM_VALUE through FLEX20_NUM_VALUE	W_RTL_ DEALINC_IT_LC_WK_A.FLEX1_NUM_VALUE through W_RTL_ DEALINC_IT_LC_WK_A.FLEX20_NUM_VALUE
Deal Income Fixed Basis	W_PDS_DEALINC_IT_LC_WK_A	DEAL_INCOME_FIXED	W_RTL_DEALINC_IT_LC_WK_A.DEAL_FIXED_COST_AMT
Deal Income Fixed Basis – Retail	W_PDS_DEALINC_IT_LC_WK_A	DEAL_RTL_INCOME_FIXED	W_RTL_DEALINC_IT_LC_WK_A.DEAL_FIXED_RTL_AMT
Deal Income Issues Based	W_PDS_DEALINC_IT_LC_WK_A	DEAL_INCOME_ISSUES	W_RTL_DEALINC_IT_LC_WK_A.DEAL_ISSUE_COST_AMT
Deal Income Issues Based - Retail	W_PDS_DEALINC_IT_LC_WK_A	DEAL_RTL_INCOME_ISSUES	W_RTL_DEALINC_IT_LC_WK_A.DEAL_ISSUE_RTL_AMT

Intercompany Margin Mapping

Data is loaded from the `IC_MARGIN.csv` file or from RMFCS. The primary data warehouse table is the week-level aggregate generated by the historical load process.

Measure	Target Table	Target Column	Data Source
Intercompany Margin	W_PDS_ICM_IT_LC_WK_A	INTERCOMPANY_MARGIN	W_RTL_ICM_IT_LC_WK_A.IC_MARGIN_AMT

Allocation Detail Mapping

Data is loaded from the `W_RTL_ALC_DETAILS_DS.dat` and `W_RTL_ALC_IT_LC_DY_FS.dat` files or from RMFCS. This data is low-level daily allocation details specifically for IPO usage.

Measure	Target Table	Target Column	Data Source
Allocation Number	W_PDS_ALC_IT_LC_DY_F	ALLOC_NO	W_RTL_ALC_DETAILS_D.ALLOC_NO
Item	W_PDS_ALC_IT_LC_DY_F	ITEM	W_PDS_PRODUCT_D.ITEM
To Location	W_PDS_ALC_IT_LC_DY_F	TO_LOC	W_PDS_ORGANIZATION_D.LOCATION
To Location Type	W_PDS_ALC_IT_LC_DY_F	TO_LOC_TYPE	W_PDS_ORGANIZATION_D.LOC_TYPE
From Location	W_PDS_ALC_IT_LC_DY_F	FROM_LOC	W_RTL_ALC_DETAILS_D.SOURCE_WH
From Location Type	W_PDS_ALC_IT_LC_DY_F	FROM_LOC_TYPE	'W'

Measure	Target Table	Target Column	Data Source
In Store Date	W_PDS_ALC_IT_LC_DY_F	IN_STORE_DATE	W_RTL_ALC_IT_LC_DY_F.IN_STORE_DATE
Transferred Qty	W_PDS_ALC_IT_LC_DY_F	TRANSFERRED_QTY	W_RTL_ALC_IT_LC_DY_F.TRANSFERRED_QTY
Received Qty	W_PDS_ALC_IT_LC_DY_F	RECEIVED_QTY	W_RTL_ALC_IT_LC_DY_F.RECEIVED_QTY
Allocated Qty	W_PDS_ALC_IT_LC_DY_F	ALLOCATED_QTY	W_RTL_ALC_IT_LC_DY_F.ALLOCATED_QTY
Prescaled Qty	W_PDS_ALC_IT_LC_DY_F	PRESCALED_QTY	W_RTL_ALC_IT_LC_DY_F.PRESCALED_QTY
Reconciled Qty	W_PDS_ALC_IT_LC_DY_F	RECONCILED_QTY	W_RTL_ALC_IT_LC_DY_F.RECONCILED_QTY
Distro Qty	W_PDS_ALC_IT_LC_DY_F	DISTRO_QTY	W_RTL_ALC_IT_LC_DY_F.DISTRO_QTY
Selected Qty	W_PDS_ALC_IT_LC_DY_F	SELECTED_QTY	W_RTL_ALC_IT_LC_DY_F.SELECTED_QTY
Cancelled Qty	W_PDS_ALC_IT_LC_DY_F	CANCELLED_QTY	W_RTL_ALC_IT_LC_DY_F.CANCELLED_QTY
Item Avg Cost	W_PDS_ALC_IT_LC_DY_F	ITEM_AVG_COST_AMT_LCL	W_RTL_ALC_IT_LC_DY_F.ITEM_AVG_COST_AMT_LCL
Pack Flag	W_PDS_ALC_IT_LC_DY_F	PACK_FLG	W_PDS_PRODUCT_D.PACK_FLG
Order Number	W_PDS_ALC_IT_LC_DY_F	ORDER_NO	W_RTL_ALC_DETAILS_D.ORDER_NO
Order Type	W_PDS_ALC_IT_LC_DY_F	ORDER_TYPE	W_RTL_ALC_DETAILS_D.ORDER_TYPE

Transfer Detail Mapping

Data is loaded from the `W_RTL_TSF_DETAILS_DS.dat` and `W_RTL_TSF_IT_LC_DY_FS.dat` files or from RMFCS. This data is low-level daily transfer details specifically for IPO usage.

Measure	Target Table	Target Column	Data Source
Transfer Number	W_PDS_TSF_IT_LC_DY_F	TSO_NO	W_RTL_TSF_DETAILS_D.TSF_NO
Item	W_PDS_TSF_IT_LC_DY_F	ITEM	W_PDS_PRODUCT_D.ITEM
To Location	W_PDS_TSF_IT_LC_DY_F	TO_LOC	W_PDS_ORGANIZATION_D.LOCATION
To Location Type	W_PDS_TSF_IT_LC_DY_F	TO_LOC_TYPE	W_PDS_ORGANIZATION_D.LOC_TYPE
From Location	W_PDS_TSF_IT_LC_DY_F	FROM_LOC	W_RTL_TSF_DETAILS_D.FROM_LOC
From Location Type	W_PDS_TSF_IT_LC_DY_F	FROM_LOC_TYPE	W_RTL_TSF_DETAILS_D.FROM_LOC_TYPE
Expected DC Date	W_PDS_TSF_IT_LC_DY_F	EXP_DC_DATE	W_RTL_TSF_DETAILS_D.EXP_DC_DATE

Measure	Target Table	Target Column	Data Source
Delivery Date	W_PDS_TSF_IT_LC_DY_F	DELIVERY_DATE	W_RTL_TSF_DETAILS_D.DELIVERY_DATE
Not After Date	W_PDS_TSF_IT_LC_DY_F	NOT_AFTER_DATE	W_RTL_TSF_DETAILS_D.NOT_AFTER_DATE
Transfer Qty	W_PDS_TSF_IT_LC_DY_F	TSF_QTY	W_RTL_TSF_IT_LC_DY_F.TSF_QTY
Fill Qty	W_PDS_TSF_IT_LC_DY_F	FILL_QTY	W_RTL_TSF_IT_LC_DY_F.FILL_QTY
Shipped Qty	W_PDS_TSF_IT_LC_DY_F	SHIP_QTY	W_RTL_TSF_IT_LC_DY_F.SHIP_QTY
Received Qty	W_PDS_TSF_IT_LC_DY_F	RECEIVED_QTY	W_RTL_TSF_IT_LC_DY_F.RECEIVED_QTY
Reconciled Qty	W_PDS_TSF_IT_LC_DY_F	RECONCILED_QTY	W_RTL_TSF_IT_LC_DY_F.RECONCILED_QTY
Distro Qty	W_PDS_TSF_IT_LC_DY_F	DISTRO_QTY	W_RTL_TSF_IT_LC_DY_F.DISTRO_QTY
Selected Qty	W_PDS_TSF_IT_LC_DY_F	SELECTED_QTY	W_RTL_TSF_IT_LC_DY_F.SELECTED_QTY
Cancelled Qty	W_PDS_TSF_IT_LC_DY_F	CANCELLED_QTY	W_RTL_TSF_IT_LC_DY_F.CANCELLED_QTY
Item Avg Cost	W_PDS_TSF_IT_LC_DY_F	TSF_AVG_COST_AMT_LCL	W_RTL_TSF_IT_LC_DY_F.TSF_AVG_COST_AMT_LCL
Item Unit Cost	W_PDS_TSF_IT_LC_DY_F	UNIT_COST_AMT_LCL	W_RTL_TSF_IT_LC_DY_F.UNIT_COST_AMT_LCL
Pack Flag	W_PDS_TSF_IT_LC_DY_F	PACK_FLG	W_PDS_PRODUCT_D.PACK_FLG

Purchase Order Detail Mapping

Data is loaded from the `ORDER_HEAD.csv` and `ORDER_DETAIL.csv` files or from Merchandising. This data is low-level daily purchase order details specifically for IPO usage. Only active orders (`STATUS = 'A'`) are exported to PDS. Orders are loaded from the CSV files at the pack component level, but separate pack item fields on the input file will be converted to new rows on this PDS export for the original pack item order quantities. IPO requires information about pack item orders to prevent understating the inventory levels at the warehouse and attempting to place new orders for packs already ordered.

Measure	Target Table	Target Column	Data Source
Date	W_PDS_PO_ONORD_IT_LC_DY_F	DAY_DATE	Fixed value using current business date
Item	W_PDS_PO_ONORD_IT_LC_DY_F	ITEM	W_RTL_PO_ONORD_IT_LC_G.PROD_IT_NUM W_RTL_PO_ONORD_IT_LC_G.PO_PACK_NUM
Location	W_PDS_PO_ONORD_IT_LC_DY_F	LOCATION	W_RTL_PO_ONORD_IT_LC_G.ORG_NUM
Order Number	W_PDS_PO_ONORD_IT_LC_DY_F	ORDER_NO	W_RTL_PO_ONORD_IT_LC_G.ORDER_NO

Measure	Target Table	Target Column	Data Source
On Order Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ONORD_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_ONORD_QTY W_RTL_PO_ONORD_IT_LC_G.PO_PACK_ONORD_QTY
On Order Cost	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ONORD_COST_AMT_LCL	W_RTL_PO_ONORD_IT_LC_G.PO_ONORD_COST_AMT_LCL W_RTL_PO_ONORD_IT_LC_G.PO_PACK_ONORD_QTY * PO_PACK_ITEM_UNIT_COST
On Order Retail	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ONORD_RTL_AMT_LCL	W_RTL_PO_ONORD_IT_LC_G.PO_ONORD_RTL_AMT_LCL W_RTL_PO_ONORD_IT_LC_G.PO_PACK_ONORD_QTY * PO_PACK_ITEM_UNIT_RETAIL
Ordered Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ORDERED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_ORDERED_QTY W_RTL_PO_ONORD_IT_LC_G.PO_PACK_ORDERED_QTY
Received Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_RECEIVED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_RECEIVED_QTY W_RTL_PO_ONORD_IT_LC_G.PO_PACK_RECEIVED_QTY
Cancelled Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_CANCELLED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_CANCELLED_QTY
Item Unit Cost	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ITEM_UNIT_COST	W_RTL_PO_ONORD_IT_LC_G.PO_ITEM_UNIT_COST W_RTL_PO_ONORD_IT_LC_G.PO_PACK_ITEM_UNIT_COST
Item Unit Retail	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ITEM_UNIT_RETAIL	W_RTL_PO_ONORD_IT_LC_G.PO_ITEM_UNIT_RETAIL W_RTL_PO_ONORD_IT_LC_G.PO_PACK_ITEM_UNIT_RETAIL
Last Received Date	W_PDS_PO_ONORD_IT_LC_DY_F	PO_LAST_RECEIVED_DT	W_RTL_PO_ONORD_IT_LC_G.PO_LAST_RECEIVED_DT
Last Received Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_LAST_RECEIVED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_LAST_RECEIVED_QTY
Prescaled Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_PRESCALED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_PRESCALED_QTY
Rounded Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_LAST_ROUNDED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_LAST_ROUNDED_QTY
Group Rounded Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_LAST_GRP_ROUNDED_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_LAST_GRP_ROUNDED_QTY
Original Replenishment Units	W_PDS_PO_ONORD_IT_LC_DY_F	PO_ORIGINAL_REPL_QTY	W_RTL_PO_ONORD_IT_LC_G.PO_ORIGINAL_REPL_QTY

Measure	Target Table	Target Column	Data Source
Estimated In Stock Date	W_PDS_PO_ONORD_IT_LC_DY_F	ESTIMATED_INSTOCK_DATE	W_RTL_PO_ONORD_IT_LC_G.ESTIMATED_INSTOCK_DATE
Pack Flag	W_PDS_PO_ONORD_IT_LC_DY_F	PACK_FLG	Fixed value of `N` or `Y` based on whether the record is for non-pack or pack item orders, respectively.

Transformations in Planning

Planning applications allow the loading of fact data at the load intersection level (such as Item and Location) but uses the data within the application at an aggregated level (called the base intersection). In MFP, though all facts are loaded at the item level, it only needs data to plan at the Subclass level. The data will be aggregated from item level to subclass level for all the configured metrics to be directly used by the application. During re-classifications (such as when one item is moved from one subclass to another subclass), after the new hierarchy details are imported into MFP it also triggers re-classification of all fact data. Re-aggregation of fact data then happens only for shared facts having different load and base intersections.

In Planning applications, fact data is grouped as dynamic fact groups based mainly on the base intersection and interface details, as defined in the Data Interface of the Application Configuration. RI and AI Foundation use a relational data model, whereas Planning applications internally use a hierarchical data model. Data from RAP, stored using the relational data model, needs to be transformed to be loaded into Planning applications. A similar approach is necessary for data coming out of planning applications to AI Foundation or external sources. These data transformations happen as part of the interfaces defined in `interface.cfg` (Interfaces Configuration File), which is a mapping of dimensions and measures from Planning applications to external system table columns. Refer to the application-specific Implementation Guides for more information about Planning Data Interfaces.

7

Implementation Tools

Review the sections below to learn about the tools and common components used within the Retail Analytics and Planning. Many of these tools are used both for initial implementation and for ongoing maintenance activities, so implementers should be prepared to transfer knowledge of these tools to the customer before completing the project.

Retail Home

One of the first places you will go in a new RAP environment is Retail Home. It serves both as the customer portal for Oracle Retail cloud applications and as a centralized place for certain common configurations, such as Customer Module Management. Module management allows implementers to quickly configure the complex batch schedules and interdependencies of RAP applications using a simplified module-based layout. Optional batch programs, such as those used for Retail Insights or AI Foundation applications, can be turned off from this tool and it synchronizes with the batch scheduler to ensure all related programs are disabled automatically.

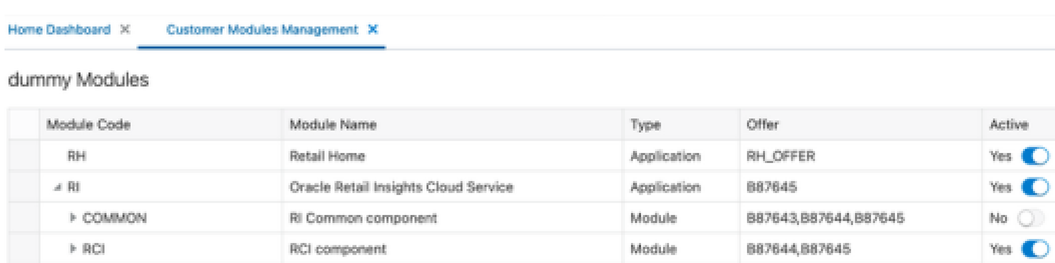
For more general information about Retail Home and the other features it provides, review the *Retail Home Administration Guide*.

Because Customer Modules are a necessary part of configuring and using a RAP environment, see the steps below for how to access this feature.

1. To access Retail Home, access the URL sent to your cloud administrator on first provisioning a new environment. It should look similar to the URL format below.

   ```
   https://{service}.retail.{region}.ocs.oraclecloud.com/{solution-customer-env}/retailhome
   ```

2. Navigate to **Settings** → **Application Administration** → **Customer Modules Management**. Confirm the table on this page loads without error and displays multiple rows of results. If an error occurs, contact Oracle Support.

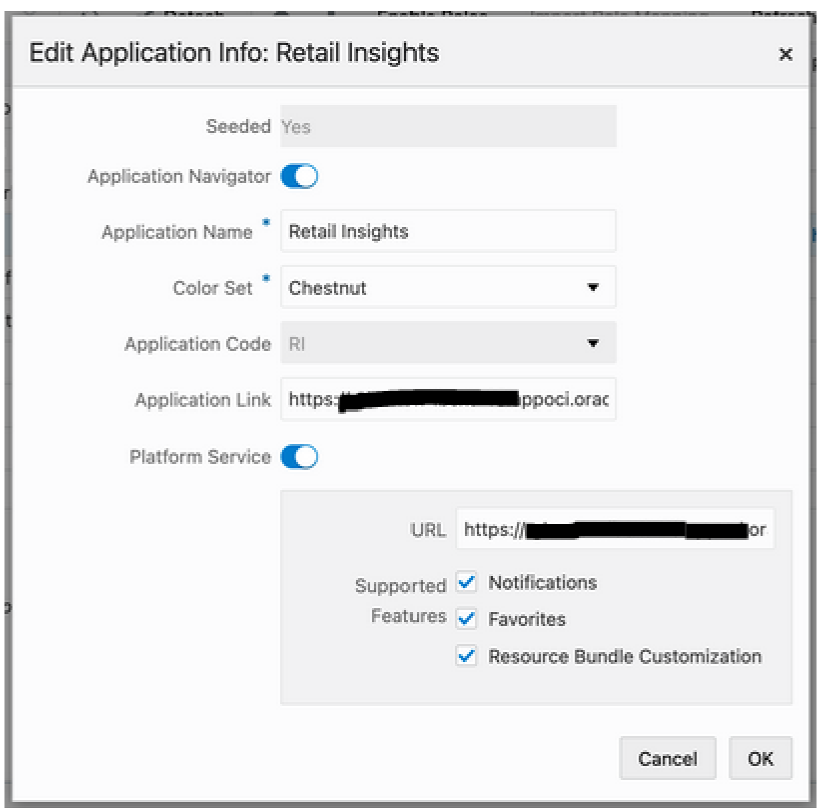
3. You may enable or disable various modules, depending on your implementation plans. For example, if you are not implementing any Retail Insights modules, then the sections for “RCI” and “RMI” can be deactivated.

> **Note:**
>
> Other components within the RI parent module may still be necessary. Detailed module requirements are described in Batch Orchestration

In addition to Customer Modules, you may also use Retail Home's Resource Bundle Customization (RBC) feature to change translatable strings in the applications to custom values. Use the steps below to verify this feature is available:

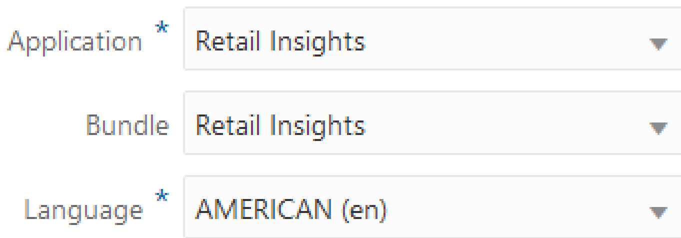
1. Navigate to **Settings** → **Application Administration** → **Application Navigator Setup**.
2. Confirm that a row already exists for each application in the platform, including Retail Insights, Retail AI Foundation Cloud Services, and Merchandise Financial Planning.
3. On Retail Insights, select the row and click **Edit**.
   a. If not enabled, change the **Platform Service** toggle to an enabled state.
   b. Check all of the boxes that appear.
   c. Enter a valid platform service **URL**.

      If your platform services URL is blank and you do not know the URL, log a Service Request to receive it from Oracle.
4. Repeat the steps above for the AI Foundation and MFP modules, if necessary.



5. Navigate to **Settings** → **Resource Bundles** → **Resource Text Strings** once the navigator and platform service setup is validated.
6. Set the following values in the dropdown menus:
   a. **Application:** Retail Insights

b. **Bundle:** Retail Insights
c. **Language:** AMERICAN (en)



7. Click the **Search** button and wait for results to return.
   - If multiple rows of results are returned, then you have successfully verified the feature is enabled and functioning properly.
   - If you receive an error, contact Oracle Support.

If you need additional details on how the RBC feature is used within each application module, refer to the product-specific documentation sets, such as the *Retail Insights Administration Guide*.

Process Orchestration and Monitoring (POM)

The Process Orchestration and Monitoring (POM) application is a user interface for scheduling, tracking, and managing both nightly as well as intraday batch jobs for applications such as RI, MFP, and AI Foundation. Two important screens from the POM application are Batch Monitoring and Batch Administration. The Batch Monitoring window provides a runtime view of the statuses and dependencies of the different batch cycles running on the current business day. Batch Administration allows you to modify the batch schedules and synchronize them with Retail Home.

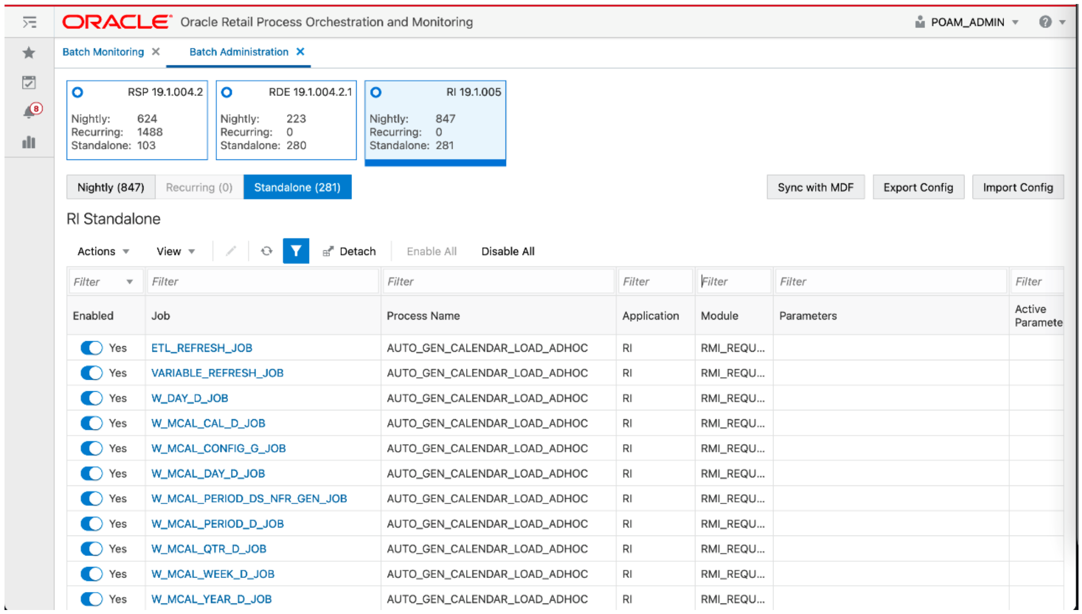
For general information about POM and the features it provides, review the *POM Implementation Guide* and *POM User Guides*.

POM and Customer Modules Management

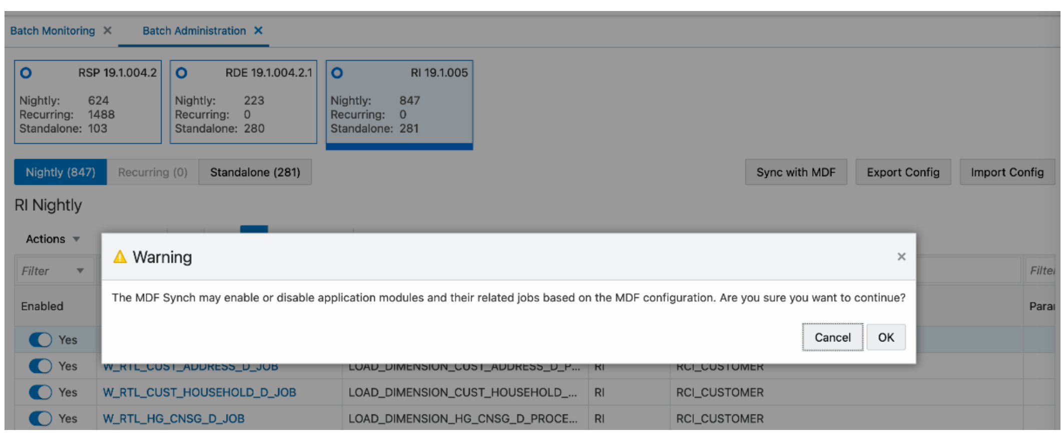
A required implementation step for RAP will be to synchronize customer modules from Retail Home to POM to set up your starting batch processes and turn off any processes you are not using. The steps below explain the general process for syncing POM and Retail Home, which are used by Retail Insights and AI Foundation applications to set up the nightly batches. They are also required for the RAP common components used by all the modules.

1. Log in to the Retail Home application as a user with the `RETAIL_HOME_ADMIN` (or `RETAIL_HOME_ADMIN_PREPROD`) user role.
2. Navigate to **Settings ->Application Administration -> Customer Modules Management**.
3. Configure your batch modules as needed, disabling any components which you do not plan to implement, then click the **Save** button to complete the setup.
4. Now log in to the POM application URL with a user granted the `BATCH_ADMINISTRATOR_JOB` or pre-prod equivalent role.
5. Navigate to **Tasks -> Batch Administration**.

6. Click on the tile named **AIF DATA <Release_#>** to view the RI and data warehouse batch jobs, which should be loaded into the table below the tiles.



7. Click the **Sync with MDF** button (above the table) and then click the **OK** button in the Warning message popup. Once clicked, the Platform Services calls are initiated between Retail Home and POM to sync the module status.

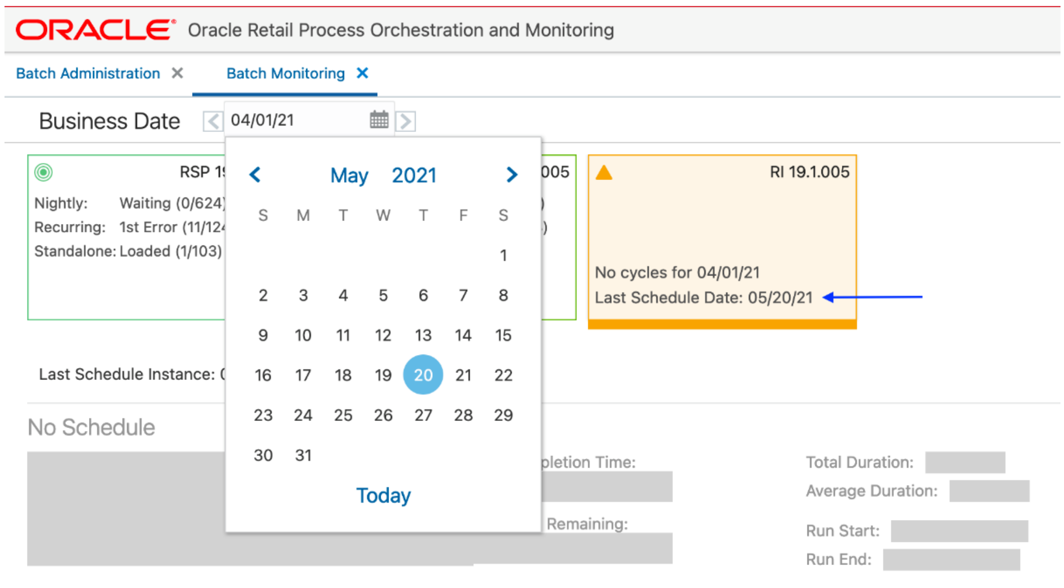


8. While the modules are synchronizing, you will see a message: 'Some features are disabled while a schedule is being synced'. Do not attempt to modify the schedule or make other changes while the sync is in progress.

9. Once the sync is complete, a JSON file with the batch schedule summary is downloaded. This file contains the current and previous status of an application and module in MDF and POM after sync. For example:
{"scheduleName":"RI","synced":true,"enabledModules":
[{"state":"MATCHED_MODULE","mdfStatus":"ENABLED","prevMdfStatusInPom":"E

NABLED","prevStatusInPom":"ENABLED","publishToPom":true,"applicationName":"RI","moduleName":"RMI_SI_ONORDER","matchedModule":true},...

10. Click the **Nightly** or the **Standalone** tab above the table and enter a filter for the Module column (based on the modules that were activated or deactivated) and press `Enter`. The jobs will be enabled or disabled based on the setup in Customer Modules Management.
11. Navigate to **Tasks** → **Batch Monitoring**. Click on the same application tile as before. If the batch jobs are not listed, change the **Business Date** option to the 'Last Schedule Date' shown on the tile.

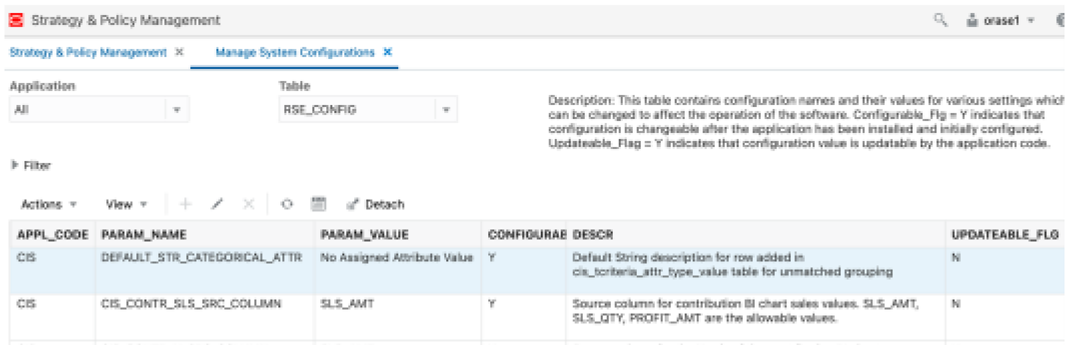


12. Once the date is changed, the batch jobs are loaded in the table. Click the **Restart Schedule** button so that module changes are reflected in the new schedule. Click **OK** on the confirmation pop-up. After a few seconds, a 'Restarted' message is displayed.
13. In the same screen, filter the Job column (for example,'W_HOUSEHOLD') to check the status of jobs. The status is either 'Loaded' or 'Disabled' based on the configuration in the Customer Modules Management screen in Retail Home.

> **Note:**
>
> A specific module in Retail Home may appear under several applications, and jobs within a module may be used by multiple processes in POM. The rule for synchronizing modules is, if a given POM job is enabled in at least one module, it will be enabled in POM (even if it is disabled in some other modules). Only when a job is not needed for any modules will it be disabled.

Control & Tactical Center

Retail Insights and AI Foundation modules make use of a centralized configuration interface named the Control & Tactical Center. From here the user can review and override the system configurations for different applications through the Manage System Configurations screen. The table can be filtered by Application and their configured tables. There is also a Description section on the right side that displays the details of the filtered table.



Here are the steps for accessing and using this feature:

1. To access the system configurations, start from the Retail Home URL sent to your cloud administrator on first provisioning a new environment. It should look similar to the URL format below.

   ```
   https://{service}.retail.{region}.ocs.oraclecloud.com/{solution-customer-env}/retailhome
   ```

2. Using the Retail Home application menu, locate the link for the Retail AI Foundation Cloud Services. Alternatively, you can directly navigate to the application using a URL similar to the format below.

   ```
   https://{service}.retail.{region}ocs.oraclecloud.com/{solution-customer-env}/orase/faces/Home
   ```

3. In the task menu, navigate to **Control & Tactical Center** → **Strategy & Policy Management**. A new window opens.

   > **Note:**
   >
   > Make sure your user has the `ADMINISTRATOR_JOB` role in OCI IAM before logging into the system.

4. Click **Manage System Configurations** in the new application screen.
5. Select an application in the dropdown menu to pick the desired set of configurations. Based on the selection, the Filter and Table options are populated with the configured columns and data. The Description section also displays the details of the selected table.

Specifically for the initial environment setup, you will be working mainly within the Retail Insights group of configuration tables. You will also use the Strategy & Policy Management interface to access the forecasting configurations needed to set up and generate forecasts for Planning applications. It is also used to manage the business policies and rules used by Lifecycle Pricing Optimization. Any required configurations in these areas will be covered later in this document.

Data Visualizer

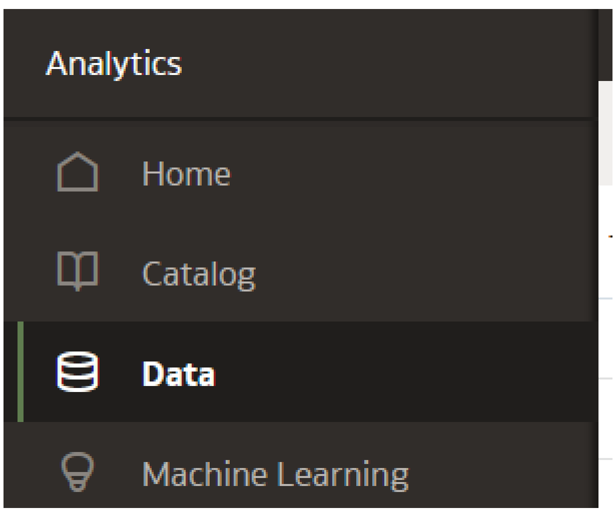
Retail Analytics and Planning implementations largely involve processing large volumes of data through several application modules, so it is important to know how to access the database to review settings, monitor load progress, and validate data tables. Database access is provided through the Oracle Data Visualization (DV) tool, which is included with all Retail Analytics and Planning environments. The URL to access the DV application will be similar to the below URL:

```
https://{analytics-service-region}/{tenant-id}/dv/?pageid=home
```

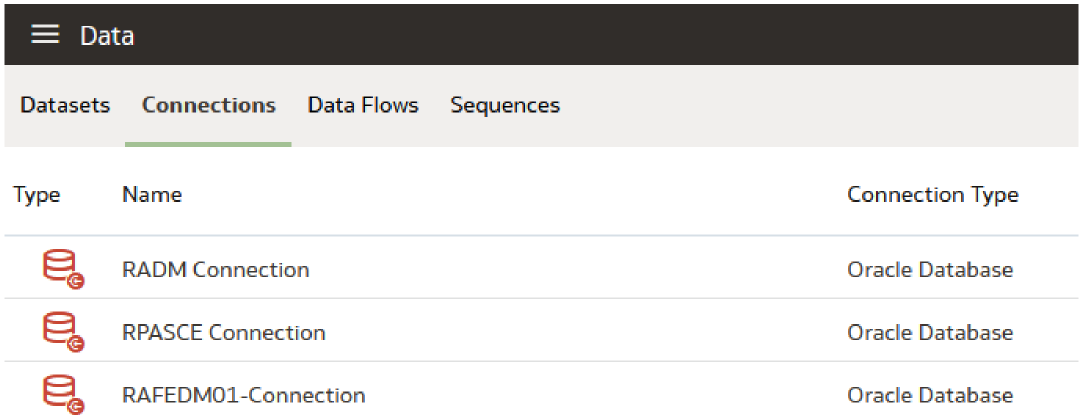
> **Note:**
>
> The best way to write ad hoc SQL against the database is through APEX. However, Data Visualizer can be used to create reusable datasets based on SQL that can be built into reports for longer term usage.

The RAP database comprises several areas for the individual application modules, but the majority of objects from RI and AIF are exposed in DV as a connection to the `RAFEDM01` database user. This user has read-only access to the majority of database objects which are involved in RI and AI Foundation implementations, as well as the tables involved in publishing data to the Planning modules. Follow the steps below to verify access to this database connection:

1. Log in to the DV application with a user that includes the `DVContentAuthor` group in OCI IAM (group names vary by cloud service; they will be prefixed with the tenant ID).
2. Expand the navigation panel using the Navigator icon in the upper left corner.

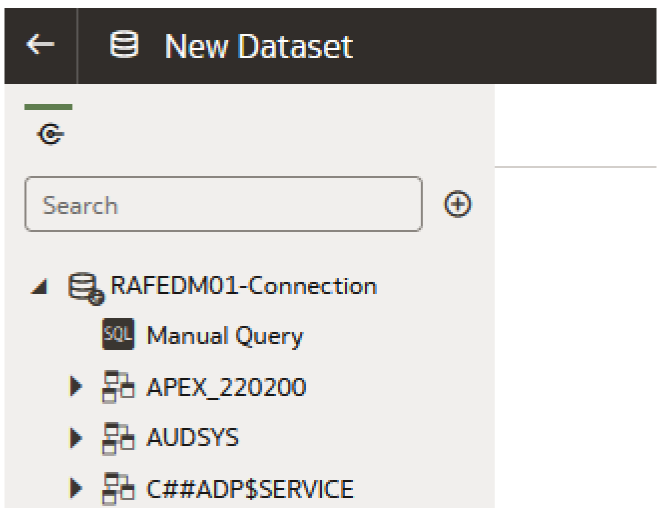


3. Click **Data** and, once the screen loads, click **Connections**. Confirm that you have a connection already available for RAFEDM01-Connection (Retail Analytics Front End Data Mart).

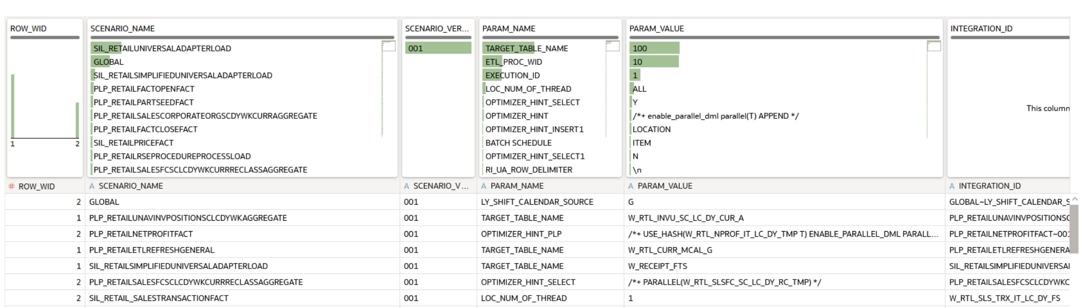


4. Click the connection. The Add Data Set screen will load using the selected connection. A list of database users are displayed in the left panel.

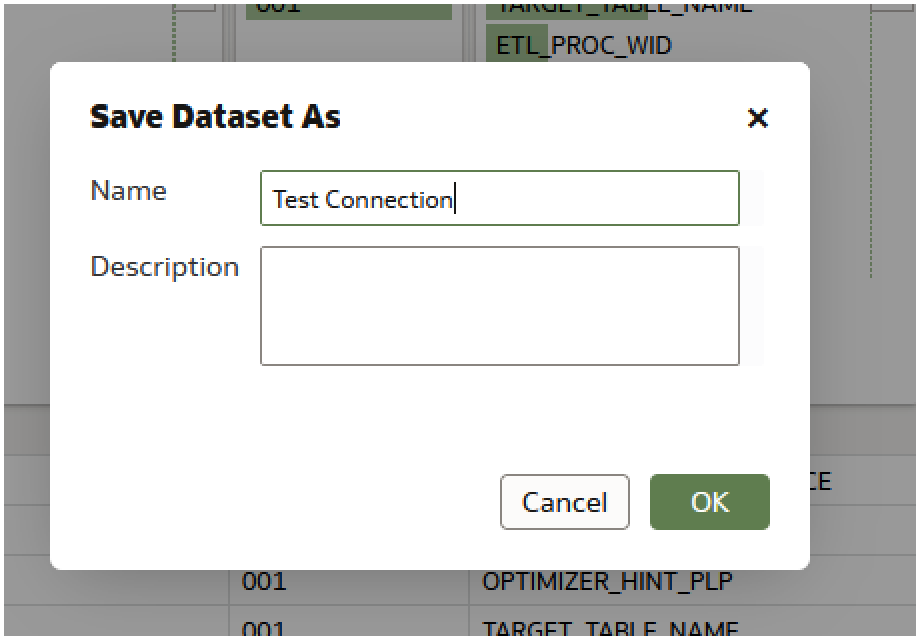
   If any errors are displayed or a password is requested, contact Oracle Support for assistance.



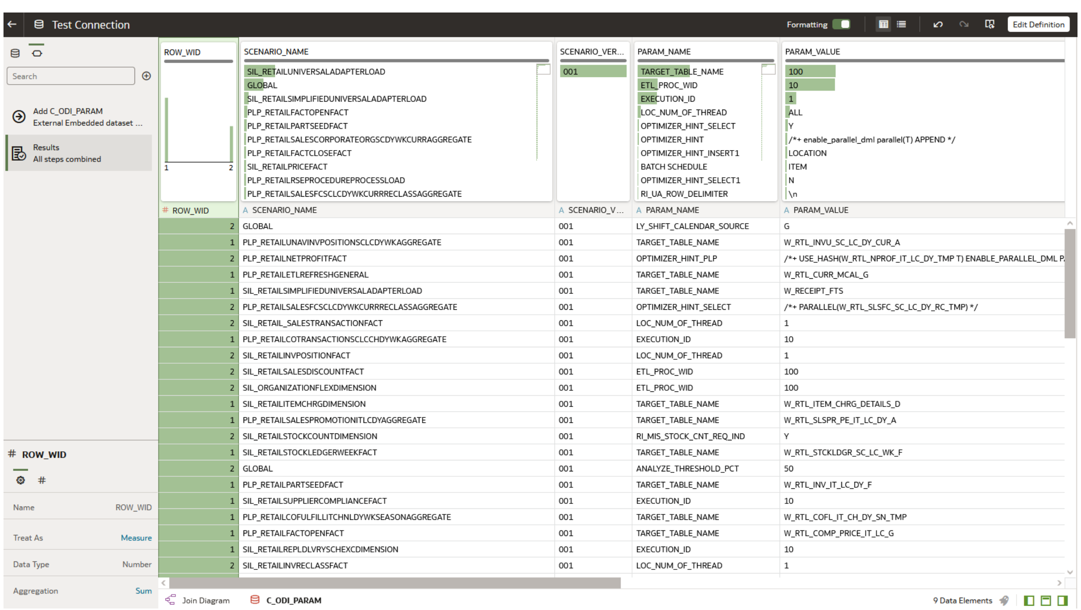
5. Expand the **RAFEDM01** user.
6. Select **C_ODI_PARAM** from the list of database tables and drag-and-drop the table name into the center of the screen.
7. Wait for the table to be analyzed and results to be displayed. Confirm that multiple rows of data are shown in the bottom panel:



8. If you are performing a one-time query that does not need to be repeated or reused, you can stop at this point. You can also add a **Manual SQL** query using the SQL object at the top of the left panel, to write simple queries on the database. However, if you want to create a reusable dataset, or expose the data for multiple users, proceed to the next steps.
9. Click the **Save** icon in the upper right corner of the screen and provide a name for the new dataset:



10. Click the table name (`C_ODI_PARAM`) at the bottom of the screen to modify the dataset further for formatting and custom fields (if desired).
11. You can format the dataset on this screen for use in DV projects. You may rename the columns, change the datatype between Measure and Attribute, create new columns based on calculated values, and extract values from existing columns (such as getting the month from a date). Refer to Oracle Analytics documentation on Dataset creation for full details. When finished, click **Create Workbook** in the upper right corner to open a new workbook with it.



Once you have verified database connectivity, you may continue on to creating more datasets and workbooks as needed. Datasets will be saved for your user and can be reused at later dates without having to re-query the database. Saved datasets can be accessed using the Data screen from the Navigator panel.

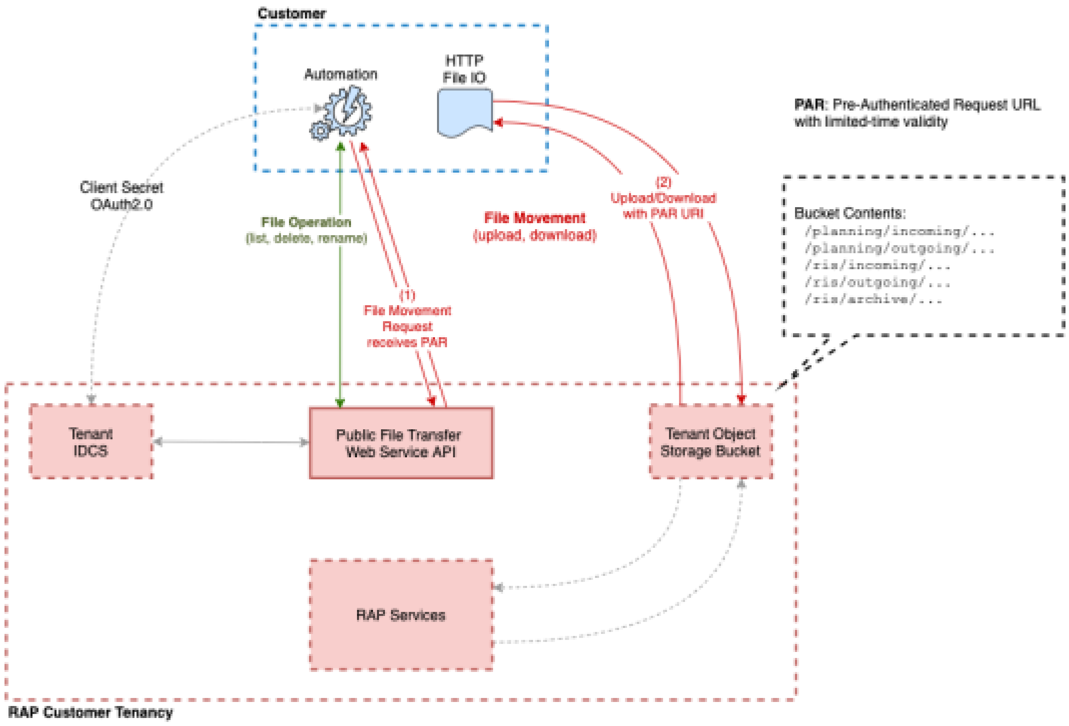
File Transfer Services

File Transfer Services (FTS) for the Retail Analytics and Planning cloud services are being made available in this release, replacing SFTP in new environments. They will allow you to manage uploading and downloading files to Oracle Cloud Infrastructure Object Storage, which is an internet-scale, high-performance storage platform that offers reliable and cost-efficient data durability.

Access to files is through a pre-authenticated request (PAR), which is a URL that requires no further authentication to upload or download files to the cloud. To retrieve a PAR, you must use the appropriate file transfer REST service. These new services will enable you to import files to and export files from Object Storage used by the solutions. The primary role of these services is to ensure that only valid external users can call the service by enforcing authorization policies. Where supported, the files can be compressed (zipped) before upload.

The general flow of activities involving FTS and an external integrating system are as follows for transferring files into our cloud service:

1. The third-party solution calls the service, requesting pre-authentication to upload files from Object Storage, including the incoming prefix and file name. On receiving the PAR, the file is uploaded using the URL included within the response. A PAR has a validity of 10 minutes; an upload can take longer than 10 minutes but after it is returned it must be started within that period.
2. The cloud service batch processing will retrieve the files from Object Storage, after they have passed an anti-virus and malware scan.
3. The batch processing will delete the file from Object Storage to ensure it is not re-processed in the next batch run. The batch processing uncompresses the file and processes the data.



To interact with FTS you must use the REST APIs provided. The table below lists the API end points for different file operations.

Operation	Method	FTS API Endpoint
Ping	GET	`{baseUrl}/services/private/FTSWrapper/ping`
List Prefixes	GET	`{baseUrl}/services/private/FTSWrapper/listprefixes`
List Files	GET	`{baseUrl}/services/private/FTSWrapper/listfiles`
Move Files	POST	`{baseUrl}/services/private/FTSWrapper/movefiles`
Delete Files	DELETE	`{baseUrl}/services/private/FTSWrapper/delete`
Request Upload PAR	POST	`{baseUrl}/services/private/FTSWrapper/upload`
Request Download PAR	POST	`{baseUrl}/services/private/FTSWrapper/download`

The `{baseUrl}` is the URL for your RAP service that is supplied to you when your service is provisioned, and can be located from Retail Home as it is also the platform service URL. Refer to the Required Parameters section for additional parameters you will need to make FTS requests.

Required Parameters

To leverage File Transfer Services, several pieces of information are required. This information is used in API calls and also inserted into automated scripts, such as the test script provided later in this document.

The below parameters are required for uploading data files to object storage.

```
BASE_URL="https://__YOUR_TENANT_BASE_URL__"
TENANT="__YOUR-TENANT_ID__"
IDCS_URL="https://_YOUR__IDCS__URL__/oauth2/v1/token"
IDCS_CLIENTID="__YOUR_CLIENT_APPID__"
IDCS_CLIENTSECRET="__YOUR_CLIENT_SECRET___"
IDCS_SCOPE="rgbu:rsp:psraf-__YOUR_SCOPE__"
```

Base URL

The substring before the first '/' in the Application URL is termed as the base URL.

Example URL: `https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com/rgbu-rap-hmcd-stg1-rsp/orase/faces/Home`

In the above URL, `BASE_URL` = `https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com`

Tenant

The string after the Base URL and before the Application URL starts would be the Tenant.

Example URL: `https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com/rgbu-rap-hmcd-stg1-rsp/orase/faces/Home`

In the above URL, `TENANT` = `rgbu-rap-hmcd-stg1-rsp`

OCI IAM URL

Your authentication URL is the one used when you first access any of your cloud services and are redirected to a login screen. You will get the base URL from the login screen and combine it with the necessary path to fetch the authentication token.

Example Base URL: `https://oci—iam-a4cbf187f29d4f41bc03fffb657d5513.identity.oraclecloud.com/`

Using the above URL, `IDCS_URL` = `https://oci—iam-a4cbf187f29d4f41bc03fffb657d5513.identity.oraclecloud.com/oauth2/v1/token`

OCI IAM Scope

The authentication scope is a code associated with the specific environment you are planning to interact with. It has a static prefix based on the application, appended with an environment-specific code and index.

Base format: `rgbu:rsp:psraf-<ENV><ENVINDEX>`

Where `<ENV>` is replaced with one of the codes in (`PRD`, `STG`) and `<ENVINDEX>` is set to 1, unless you have multiple staging environments, and then the index can be 2 or greater. For these applications (that is, RI and AIF) use the `rsp` code. For other applications, the code is `rpas`.

To determine this information, look at the URL for your environment, such as:

```
https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com/rgbu-rap-hmcd-stg1-rsp/
orase/faces/Home
```

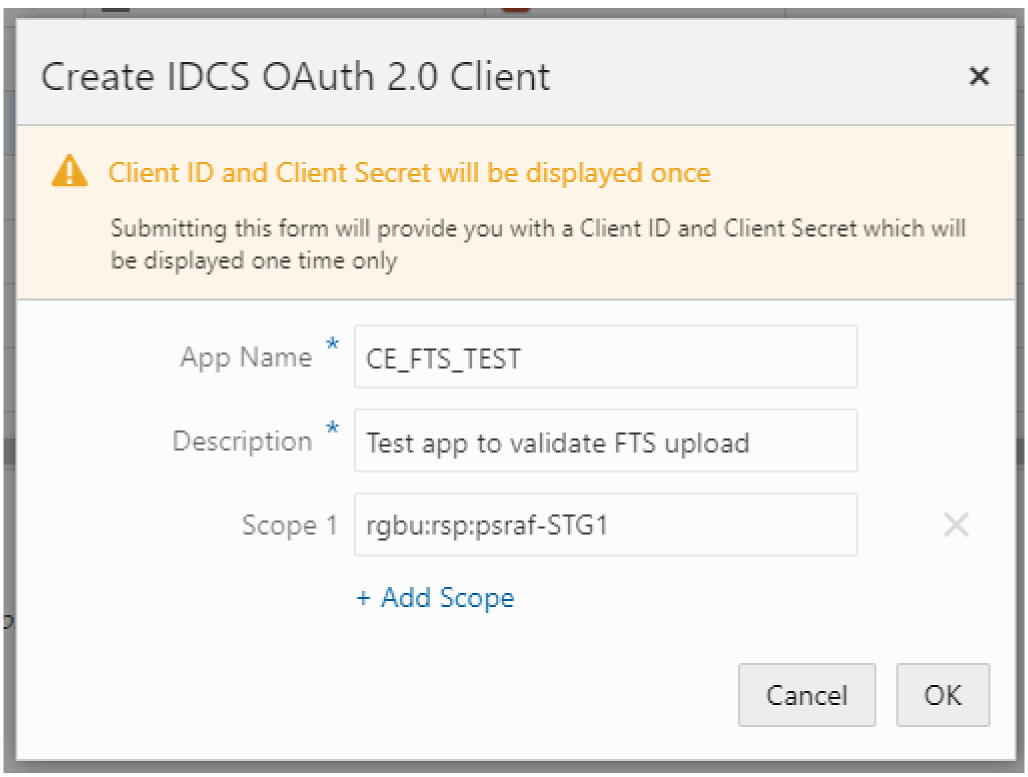
In the tenant string, you can see the code `stg1`. This can be added to your scope string (ensuring it is in uppercase characters only).

```
IDCS_SCOPE = rgbu:rsp:psraf-STG1
```

Client ID and Secret

The client ID and secret are authentication keys generated for your specific connection and must be passed with every request. Retail Home provides an interface to get these values when you login with a user having the `PLATFORM_SERVICES_ADMINISTRATOR` group.

1. Navigate to the Manage OAuth Clients screen from the **Settings** menu, under **Application Administration**.
2. Click the plus (**+**) icon to create a new OAuth 2.0 client.
3. Enter the requested details in the window. The application name should be unique to the connection you are establishing and cannot be used to generate multiple client ID/secret pairs.

   The application name cannot be re-used for multiple requests. It also cannot contain spaces. The scope should be the string previously established in OCI IAM Scope. The description is any value you wish to enter to describe the application name being used.
4. Click **OK** to submit the form and display a new popup with the client ID and secret for the specified Application Name. Do NOT close the window until you have captured the information and verified it matches what is shown on screen. Once you close the window, you cannot recover the information and you will need to create a new application.

MFP Example

To determine `IDCS_SCOPE`, refer to the tenant string portion (for example, `rgbu-rap-cust-`**`stg1-mfpscs`**) of your cloud service URL (for example,`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-`**`stg1-mfpscs`**`/rpasceui/`)

Planning Cloud Service URL Patterns	SCOPE
`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-stg1-mfpscs/rpasceui/`	`rgbu:rpas:psraf-MFPSCS-STG1`
`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-`**`stg1-rpasce`**`/rpasceui/`	`rgbu:rpas:psraf-RPASCE-STG1`

Based on the tenant string (for example, `rgbu-rap-cust-stg1-mfpscs`), the environment index is `stg1` and the application is `mfpscs`. For this combination, the IDCS scope will look like the configuration below (ensuring it is in uppercase characters only):

```
IDCS_SCOPE = rgbu:rpas:psraf-MFPSCS-STG1
```

Create the OAuth Client in Retail Home with the following parameters:

- **App Name:** `MFP_STG1`
- **Description:** `FTS for MFP on STG1`
- **Scope 1:** `rgbu:rpas:psraf-MFPSCS-STG1`

This generates an OAuth Client with details like this:

- **Oauth client:**
- **App Name:** MFP_STG1
- **Client Id:** MFP_STG1_APPID
- **Client Secret:** 6aae7818-309b-4e7a-874e-f26356a675b1

You will need to capture Client Id and Client Secret. So set the FTS script variables as follows:

```
BASE_URL="https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com"
TENANT="rgbu-rap-hmcd-stg1-mfpscs"
IDCS_URL="https://oci—iam-
a4cbf187f29d4f41bc03fffb657d5513.identity.oraclecloud.com/oauth2/v1/token"
IDCS_CLIENTID="MFP_STG1_APPID"
IDCS_CLIENTSECRET="6aae7818-309b-4e7a-874e-f26356a675b1"
IDCS_SCOPE="rgbu:rpas:psraf-MFPSCS-STG1"
```

IPO Example

To determine the `IDCS_SCOPE`, refer to the tenant string portion (for example, `rgbu-rap-cust-stg1-ipocs`) of your cloud service URL (for example, `https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-stg1-ipo/rpasceui/`)

Planning Cloud Service URL Patterns	SCOPE
`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-`**`stg1-ipocs`**`/rpasceui/`	`rgbu:rpas:psraf-IPOCS-STG1`
`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-`**`stg1-rpasce`**`/rpasceui/`	`rgbu:rpas:psraf-RPASCE-STG1`

Based on the tenant string (for example, `rgbu-rap-cust-stg1-ipocs`), the environment index is `stg1` and the application is `ipocs`. For this combination, the IDCS scope will look like the configuration below (ensuring it is in uppercase characters only):

```
IDCS_SCOPE = rgbu:rpas:psraf-IPOCS-STG1
```

Create the OAuth Client in Retail Home with the following parameters:

- **App Name:** `IPOCS_STG1`
- **Description:** `FTS for IPOCS on STG1`
- **Scope 1:** `rgbu:rpas:psraf-IPOCS-STG1`

This generates an OAuth Client with details like this:

- **Oauth client:**
- **App Name:** IPOCS_STG1
- **Client Id:** IPOCS_STG1_APPID
- **Client Secret:** 6aae7818-309b-4e7a-874e-f26356a675b1

You will need to capture Client Id and Client Secret. So set the FTS script variables as follows:

```
BASE_URL="https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com"
TENANT="rgbu-rap-hmcd-stg1-ipocs"
IDCS_URL="https://oci—iam-
a4cbf187f29d4f41bc03fffb657d5513.identity.oraclecloud.com/oauth2/v1/token"
IDCS_CLIENTID="IPOCS_STG1_APPID"
IDCS_CLIENTSECRET="6aae7818-309b-4e7a-874e-f26356a675b1"
IDCS_SCOPE="rgbu:rpas:psraf-IPOCS-STG1"
```

AP Example

To determine the `IDCS_SCOPE`, refer to the tenant string portion (for example, `rgbu-rap-cust-stg1-apcs`) of your cloud service URL (for example, `https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-stg1-apcs/rpasceui/`)

Planning Cloud Service URL Patterns	SCOPE
`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-`**`stg1-apcs`**`/rpasceui/`	`rgbu:rpas:psraf-APCS-STG1`

Planning Cloud Service URL Patterns	SCOPE
`https://rap.retail.us-ashburn-1.ocs.oraclecloud.com/rgbu-rap-cust-`**`stg1-rpasce`**`/rpasceui/`	`rgbu:rpas:psraf-RPASCE-STG1`

Based on the tenant string (for example, `rgbu-rap-cust-stg1-apcs`), the environment index is `stg1` and the application is `apcs`. For this combination, the IDCS scope will look like the configuration below (ensuring it is in uppercase characters only):

```
IDCS_SCOPE = rgbu:rpas:psraf-APCS-STG1
```

Create the OAuth Client in Retail Home with the following parameters:

- **App Name:** `AP_STG1`
- **Description:** `FTS for AP on STG1`
- **Scope 1:** `rgbu:rpas:psraf-AP-STG1`

This generates an OAuth Client with details like this:

- **Oauth client:**
- **App Name:** AP_STG1
- **Client Id:** AP_STG1_APPID
- **Client Secret:** 6aae7818-309b-4e7a-874e-f26356a675b1

You will need to capture Client Id and Client Secret. So set the FTS script variables as follows:

```
BASE_URL="https://rap.retail.eu-frankfurt-1.ocs.oraclecloud.com"
TENANT="rgbu-rap-hmcd-stg1-apcs"
IDCS_URL="https://oci—iam-
a4cbf187f29d4f41bc03fffb657d5513.identity.oraclecloud.com/oauth2/v1/token"
IDCS_CLIENTID="AP_STG1_APPID"
IDCS_CLIENTSECRET="6aae7818-309b-4e7a-874e-f26356a675b1"
IDCS_SCOPE="rgbu:rpas:psraf-AP-STG1"
```

Common HTTP Headers

Each call to FTS should contain the following HTTP headers:

```
Content-Type: application/json
Accept: application/json
Accept:-Language: en
Authorization: Bearer {ClientToken}
```

The `{ClientToken}` is the access token returned by OCI IAM after requesting client credentials. This is refreshed periodically to avoid authentication errors.

Retrieving Identity Access Client Token

The access client token is returned from a POST call to the OCI IAM URL, provided at provisioning, along with the following:

Headers
`Content-Type: application/x-www-form-urlencoded`
`Accept: application/json`
`Authorization: Basic {ociAuth}`

`ociAuth` is the base64 encoding of your `{clientId}:{clientSecret}`

Data (URLEncoded)
`grant_type=client_credentials`
`scope=rgbu :rpas :psraf-{environment}`

FTS API Specification

An example shell script implementing these API calls can be found in Sample Public File Transfer Script for Planning Apps and Sample Public File Transfer Script for RI and AIF. The sample script will require all of the parameters discussed so far in this chapter to be added to it before it can be used to issue FTS commands. Refer to the table below for a more detailed list of services available in FTS.

> **Note:**
>
> The `baseUrl` in these examples is not the same as the `BASE_URL` variable passed into cURL commands. The `baseUrl` for the API itself is the hostname and tenant, plus the service implementation path (for example, `RetailAppsPlatformServices`). The sample scripts provided in the appendix show the full path used by the API calls.

Ping	Returns the status of the service, and provides an external health-check.
Method	GET
Endpoint	`{baseUrl}/services/private/FTSWrapper/ping`
Parameters	Common headers
Request	None
Response	`{ appStatus:200 }` The appStatus code follows HTTP return code standards.

List Prefixes	Returns a list of the known storage prefixes. These are analogous to directories, and are restricted to predefined choices per service.
Method	GET
Endpoint	`{baseUrl}/services/private/FTSWrapper/listprefixes`

Parameters	Common headers
Request	None
Response	A JSON array of strings containing the known prefixes.

List Files	Returns a list of the file within a given storage prefix.
Method	GET
Endpoint	`{baseUrl}/services/private/FTSWrapper/listfiles`
Parameters	Common headers
Request	Query parameters `(…/listfiles?{parameterName})` that can be appended to the URL to filter the request: `prefix` – the storage prefix to use `contains` – files that contain the specified substring `scanStatus` – file status returned by malware/antivirus scan `limit` – control the number of results in a page `offset` – page number `sort` – the sort order key
Response	A JSON `resultSet` containing array of files. For each file, there is metadata including: name, size, created and modified dates, scan status and date, scan output message.

Move Files — Moves one or more files between storage prefixes, while additionally allowing the name to be modified

Method	GET
Endpoint	`{baseUrl}/services/private/FTSWrapper/movefiles`
Parameters	Common headers
Request	An array of files containing the current and new storage prefixes and file names, as shown below.

```
{"listOfFiles": [
    {"currentPath":
        { "storagePrefix": "string",
            "fileName": "string"},
            "newPath": {
                "storagePrefix": "string",
                "fileName": "string"
            }
        }
    ]
}
```

Response	HTTP 200, request succeeded; HTTP 500, an error was encountered.

Delete Files	Deletes one more or files
Method	DELETE
Endpoint	`{baseUrl}/services/private/FTSWrapper/delete`

Parameters	Common headers
Request	A JSON array of files to be deleted. One or more pairs of `storagePrefix` and `filename` elements can be specified within the array. `{"listOfFiles": [ { "storagePrefix": "string", "fileName": "string" } ] }`
Response	A JSON array of each file deletion attempted and the result.

```
{"listOfFiles":
    [
        {
            "storagePrefix": "string",
             "fileName": "string"
        }
    ]
}
```

Request Upload PAR	Request PAR for uploading one or more files
Method	POST
Endpoint	`{baseUrl}/services/private/FTSWrapper/upload`
Parameters	Common headers
Request	A JSON array of files to be uploaded. One or more pairs of `storagePrefix` and `filename` elements can be specified within the array.
Response	A `parList` containing an array containing elements corresponding to the request including the PAR `accessUri` and name of file.

```
{ "listOfFiles":
     [
        {
            "storagePrefix": "string",
            "fileName": "string"
        }
    ]
}
```

Request Download PAR	Request PARs for downloading one or more files
Method	POST
Endpoint	`{baseUrl}/services/private/FTSWrapper/download`
Parameters	Common headers

Request	A JSON array of files to be downloaded. One or more pairs of storagePrefix and filenames can be specified within the array. `{ "listOfFiles": [ { "storagePrefix": "string", "fileName": "string" } ] }`
Response	A `parList` containing an array containing elements corresponding to the request including the PAR `accessUri` and name of file.

FTS Script Usage

Assuming you have taken the sample script and named it `file_transfer.sh` on a Unix system, you may use commands like those below to call the APIs.

Upload Files

For RAP input files (excluding direct-to-RPASCE input files) the input files must be uploaded to the object storage with a prefix of `ris/incoming`.

- **Prefix:** `ris/incoming`
- **File Name:** `RI_RMS_DATA.zip`
- **Command:**

```
sh file_transfer.sh uploadfiles ris/incoming RI_RMS_DATA.zip
```

Download Files

For RAP output files (excluding direct-from-RPASCE output files) the files must be downloaded from the object storage with a prefix of `ris/outgoing`.

- **Prefix:** `ris/outgoing`
- **File Name:** `cis_custseg_exp.csv`
- **Command:**

```
sh file_transfer.sh downloadfiles ris/outgoing cis_custseg_exp.csv
```

Download Archives

For RAP files that are automatically archived as part of the batch process, you have the ability to download these files for a limited number of days before they are erased (based on the file retention policy in your OCI region). Archive files are added to sub-folders so the steps are different from a standard download.

1. POM job logs will show the path to the archive

   For example: `incoming-10072022-163233/RAP_DATA.zip`

2. Create the directory in your local server matching the archive name.

   For example: `mkdir incoming-10072022-163233`

3. Use the `downloadfiles` command with the `ris/archive` prefix and the file path as the sub-folder and filename together.

   For example: `sh file_transfer.sh downloadfiles ris/archive incoming-10072022-163233/RAP_DATA.zip`

Analytics Publisher

The Analytics Publisher component of Oracle Analytics is available for reporting and export file generation where the data needs to be written into a specific template or layout and must be delivered to other sources such as email or Object Storage (OS). SFTP is no longer available, so if you have previously used SFTP as the report delivery method, you will now use OS.

Configuring Burst Reports for Object Storage

After configuring the Publisher data model, as you build your burst query by navigating to the bursting section, you need to be aware of some of the key parameters to be supplied to the burst query that are specific to Object Storage.

Sample Bursting Query for object storage as the delivery channel:

```
select 0 KEY,
'<template_name>' TEMPLATE,
'RTF' TEMPLATE_FORMAT,
'en-US' LOCALE,
'PDF' OUTPUT_FORMAT,
'OBJECTSTORAGE' DEL_CHANNEL,
'<output_name>' OUTPUT_NAME,
'OS' PARAMETER1,
'<prefix>'PARAMETER2,
'<file_name>' PARAMETER3
FROM "Retail Insights As-Is"
```

Key parameters that are specific to object storage are:

- `DEL_CHANNEL` — This needs to be keyed in as `OBJECTSTORAGE`.
- `PARAMETER1` — Use `OS` as the parameter value for this, because this is the preconfigured server name. The value for this should be kept as `OS` and should not be changed.
- `PARAMETER2` — Prefix under the object storage bucket where the file will be uploaded. The prefix should not start or end with a `/` character; for example `ris/outgoing` is a possible prefix to use.
- `PARAMETER3` — File name. The file will be created under the prefix; for example a filename of `order_list` with type as PDF will be created as `ris/outgoing/order_list.pdf`

Delivering Scheduled Reports through Object Storage

If the report is not using a bursting definition then, while adding the destination for the reports delivery when you schedule a report job, you will need the following set of inputs that are required to push the file to object storage.

- **Server** – The server is preconfigured as `OS` for any tenant. `OS` must always be selected.
- **Prefix** – The prefix under the object storage bucket where the file will be uploaded.
- **File Name** – The file name with which the scheduled report output will be delivered to the object storage.

The way you set up these inputs is the same as when using the bursting option. For additional details on how to set up reports delivery through object storage, refer to Set Output Options in Oracle Cloud Visualizing Data and Building Reports in Oracle Analytics Cloud.

Downloading Reports from Object Storage

Once the reports are sent to object storage, use the createPar service to get a link to download the files. This service is available using a Retail Home API, which generates a Pre-authenticated Request (PAR) to download the file. The PAR is not accessible using File Transfer Services (FTS) commands; you must directly download the file by first calling the API and then pulling the data down from the returned URL.

For details on the service API, refer to the Retail Home Administration Guide Appendix B.

Sample message body when calling the Retail Home service:

```
{
  "disUrl": "https://rgbu.gbua.ocs.oc-test.com",
  "name": "Productlist",
  "dateExpires": "2024-01-31",
  "objectName": "ris/incoming/Productlist.pdf"
}
```

The name should match the file name (without extension) that was specified in `PARAMETER3` in the bursting query. The `objectName` is the combination of the prefix, file name, and file extension (all specified in the bursting query definition) that defines the full Object Storage path to the file. Although the `OUTPUT_FORMAT` value in the bursting query is uppercase (for example, `PDF`), the file extension used for the `objectName` will be lowercase (for example, `pdf`). The value used for `dateExpires` must be greater than the current date, but not more than 365 days in the future.

When making the request, you must also add a context type header with a value of `application/json` or you will not receive a response. If the requested file is found, then you will get a response like the following:

```
{
  "status": "Success",
  "id": "B0ji5Vir/nDfUQVaFHNhYVjgHoRPO8ZnFjTUPcQIyXcEtY8HUoqeJNsdyFzqreqv:ris/
incoming/Productlist.pdf",
  "url": "https://objectstorage.us-phoenix-1.oraclecloud.com/p/
X6BRpziLKQ3xoRcSToi68L31NHxm2rhTc2lbrTpvmWm9vIpCVWNiC63tYTCWgxYW/n/
oraclegbudevcorp/b/
cds_gbua_cndevcorp_rgbu_rgbu_tpn24ouxsgftuy5qh4te_RIRSP_STG5009_1/o/ris/
incoming/Productlist.pdf"
}
```

The URL can be passed into any method you would normally use to download a file, such as a `wget` command.

Application Express (APEX)

The Retail Analytics and Planning includes a dedicated instance of Application Express (APEX) that is pre-configured for read-only database access to RAP tables and views. APEX is managed as a part of the Innovation Workbench (IW) module that includes APEX and Data Studio. You can access APEX from a task menu link in the AI Foundation Cloud Services interface, or by navigating directly to the ORDS endpoint like below:

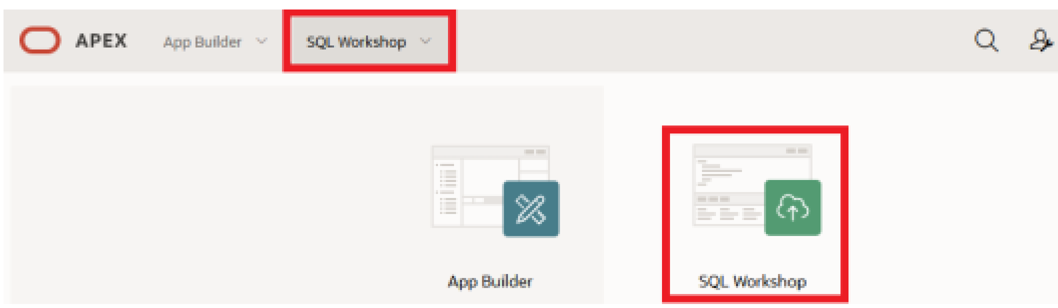
```
https://{base URL}/{solution-customer-env}/ords
```

For example:

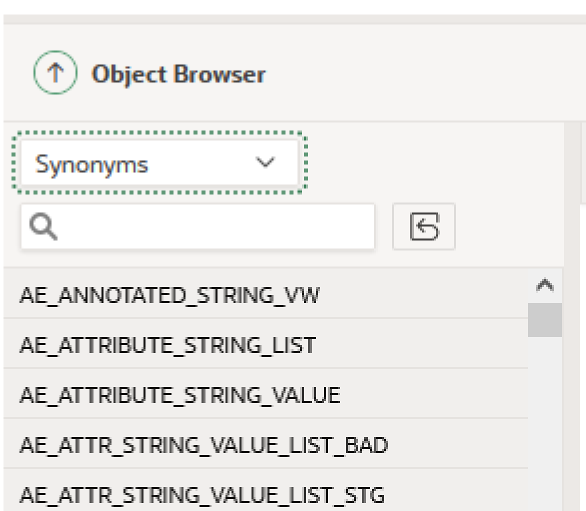
```
https://ocacs.ocs.oc-test.com/nrfy45ka2su3imnq6s/ords/
```

For first time setup of the administrator user account for APEX, refer to the *RAP Administration Guide*.

After you are logged into APEX, click the **SQL Workshop** icon or access the **SQL Workshop** menu to enter the SQL Commands screen. This screen is where you will enter SQL to query the RAP database objects in the RI and AI Foundation schemas.



To see the list of available objects to query, access the **Object Browser**. All RI and AI Foundation objects are added as synonyms, so select that menu option from the panel on the left.



If you do not see any RI tables in the synonym list, then you may need to run a set of ad hoc jobs in POM to expose them. Run the following two programs from the AI Foundation schedule's standalone job list:

- RADM_GRANT_ACCESS_TO_IW_ADHOC_JOB
- RABE_GRANT_ACCESS_TO_IW_ADHOC_JOB

Once the jobs execute successfully, start a new session of APEX and navigate back to the list of Synonyms in the Object Browser screen to confirm the table list is updated.

Database Access Levels

Your ability to query and interact with AIF database objects depends on the environment you are working with (Production or Non-Production). The following grants and privileges are given to end users in APEX specifically for tables and views in the RI and AIF database users.

Database Grant	Non-Prod	Prod
Insert (Staging Tables)	Yes	Yes
Insert (Internal Tables)	Yes	No
Update (Staging Tables)	Yes	Yes
Update (Internal Tables)	Yes	No
Delete (Staging Tables)	Yes	Yes
Delete (Internal Tables)	Yes	No
Truncate	No	No
Create Object	No	No
Alter Object	No	No
Drop Object	No	No

A staging table is generally any table that receives data from an outside source, which can include flat files, direct integration between two Oracle solutions, or web services. For example, `W_PRODUCT_DS` is the staging table in the `RADM01` schema that receives product information. Internal tables include the target tables where staged data will be moved to for the applications to read from during normal operations, temporary (`TMP`) tables, and configuration tables. There are a few exceptions made for application objects that must not be altered, but all tables that contain data you should need to interact with will be accessible in non-prod environments. `W_PRODUCT_D` is an example of a target table for the `W_PRODUCT_DS` staging table.

Postman

For automated API calls, Postman is the preferred way to interact with POM in order to call ad hoc processes and perform data load activities. The steps below explain how to configure Postman for first-time use with POM.

> **Note:**
>
> POM versions earlier than v21 allow Basic Authentication, while v21+ requires OAuth 2.0 as detailed below.

1. As a pre-requisite, retrieve the Client ID and Client Secret from Retail Home's 'Create IDCS OAuth 2.0 Client'. Refer to the *Retail Home Administration Guide* for complete details on retrieving the Client ID and Client Secret info if you have not done it before.
2. In the Postman application, click **New->HTTP Request**.
3. Set Request Type as `POST` and set the Request URL (Example for RI schedule):

```
https://<Region-LB>/<POM-Subnamespace>/ProcessServices/services/private/
executionEngine/schedules/RI/execution
```

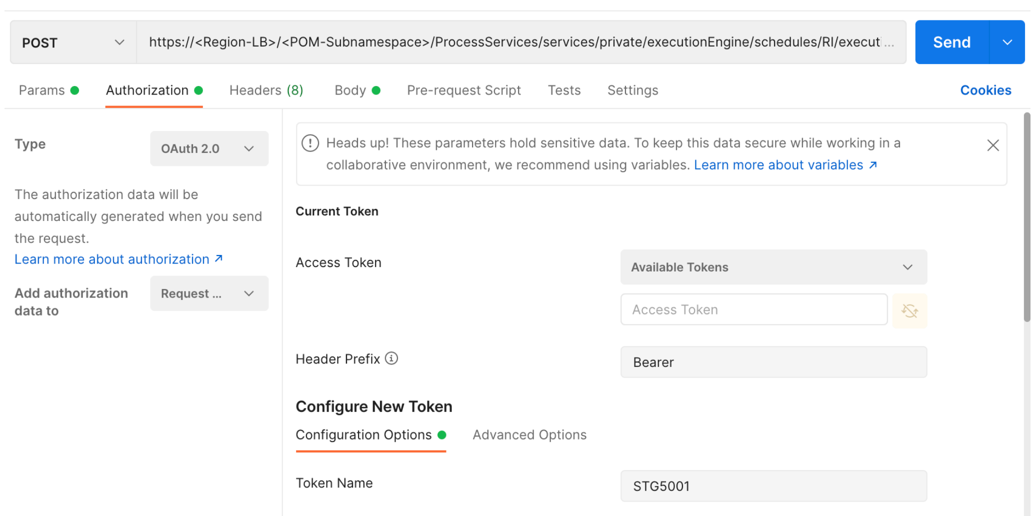
For example:

```
https://home.retail.us-region-1.ocs.oc-test.com/rgbu-common-rap-prod-pom/
ProcessServices/services/private/executionEngine/schedules/RI/execution
```

4. Perform the following steps before sending the POST request to retrieve your authentication token:
   a. Authorization Tab:

      Type: `OAuth 2.0`

      Add authorization data to: `Request Headers`



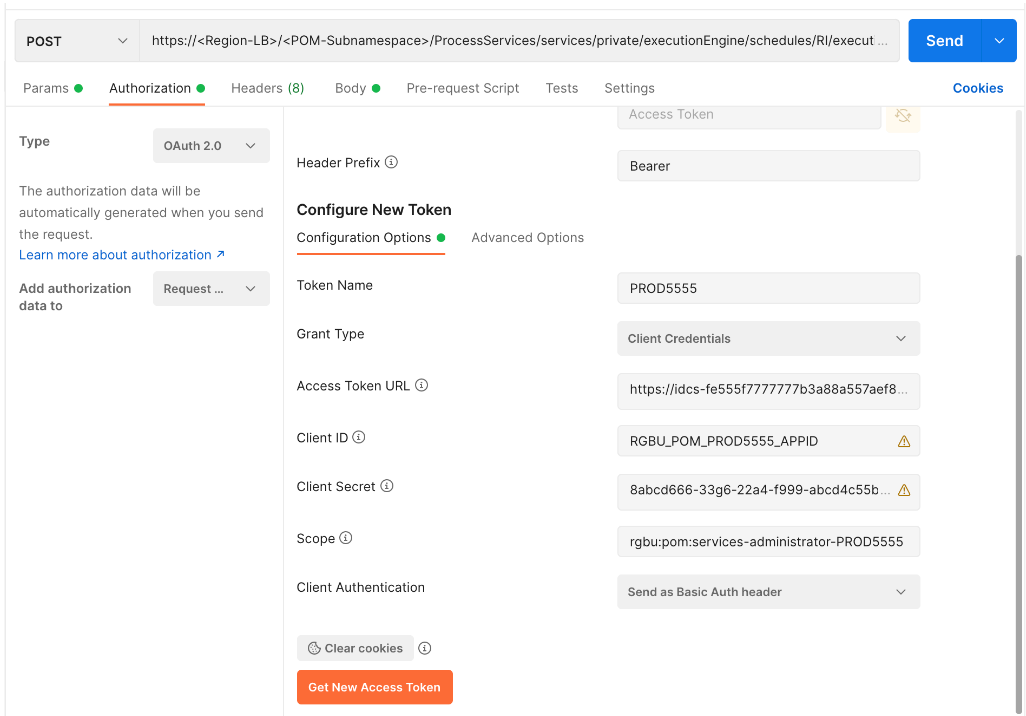
   b. Configure the New Token:
      i. **Token Name:** `PROD5555`
      ii. **Grant Type:** `Client Credentials`
      iii. **Access Token URL:** https://<Customer-OCI_IAM>/oauth2/v1/token

         For example: `https://oci—iam-fe5f77f8a44.identity.c9dev.oc9dev.com/oauth2/v1/token`
      iv. **Client ID:** A unique "API Key" generated when registering your application in the Identity Cloud Services admin console.
      v. **Client Secret:** A private key similar to a password that is generated when registering your application in the Identity Cloud Services admin console.

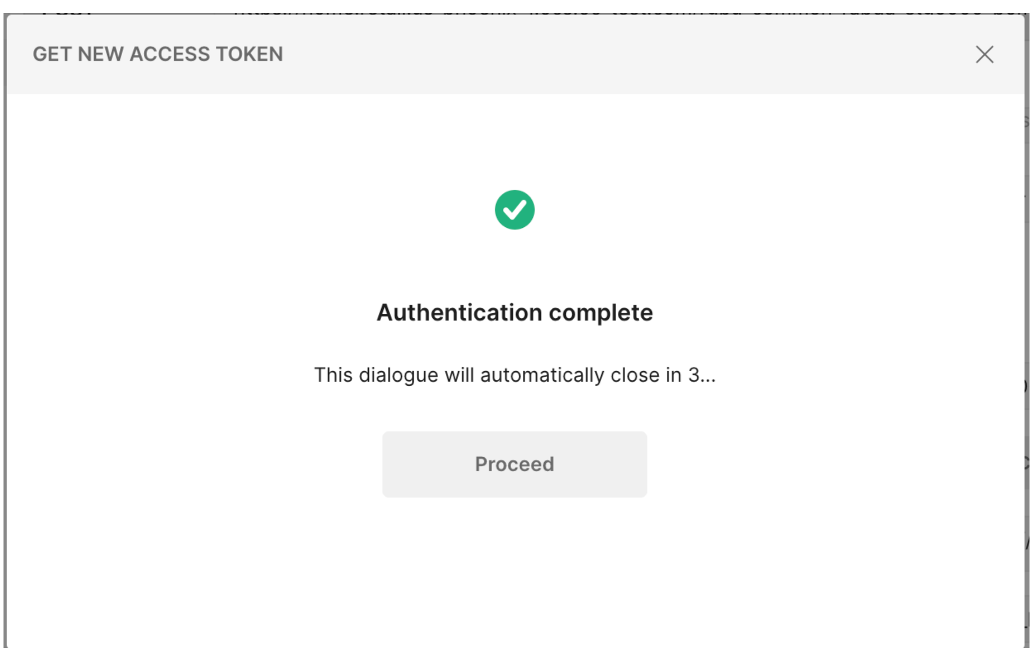
vi. **Scope:** `rgbu:pom:services-administrator-<Env-INDEX>`

For example: `rgbu:pom:services-administrator-PROD5555`

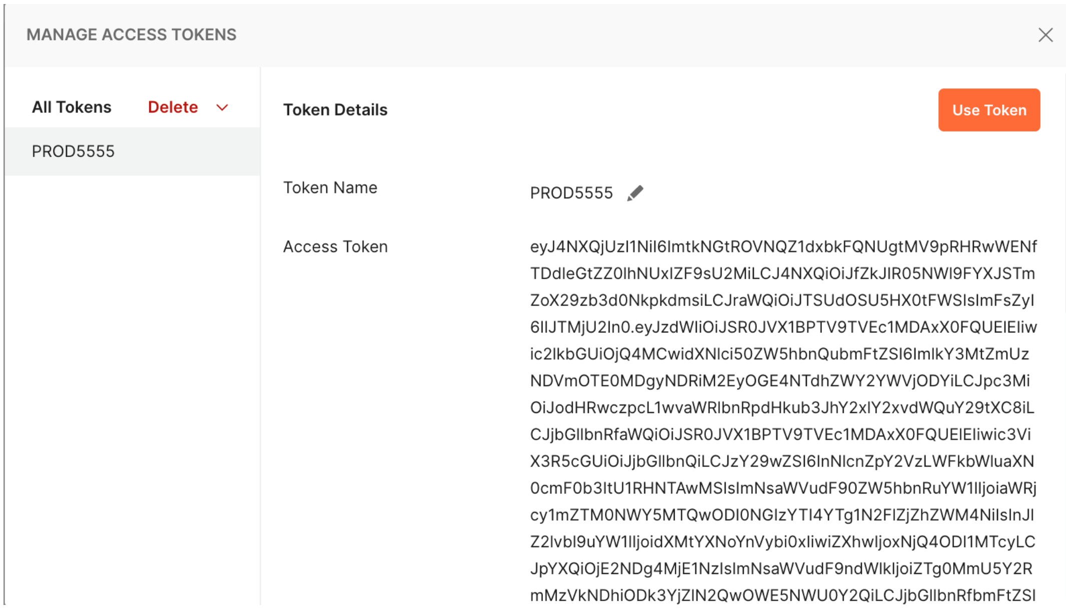
vii. **Client Authentication:** `Send as Basic Auth header`



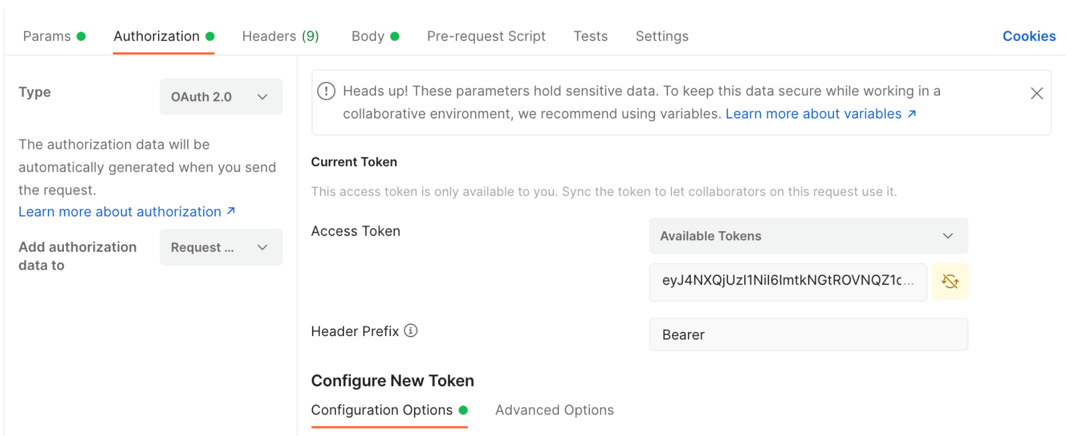
c. Click the **Get New Access Token** button.



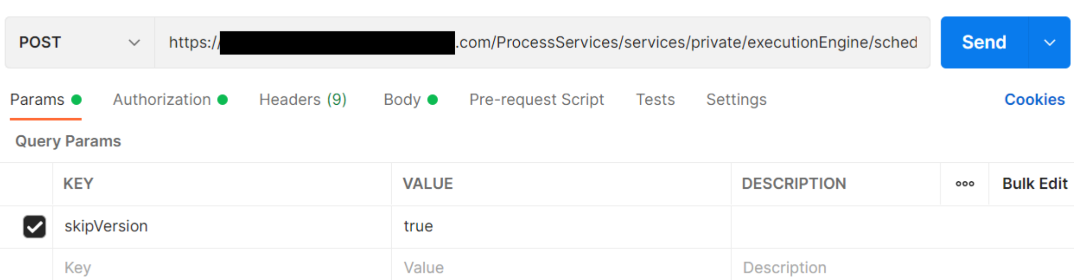
5. The Access Token is displayed in the **MANAGE ACCESS TOKENS** pop-up window. Click the **Use Token** button.



6. The generated token is populated in the **Access Token** section.



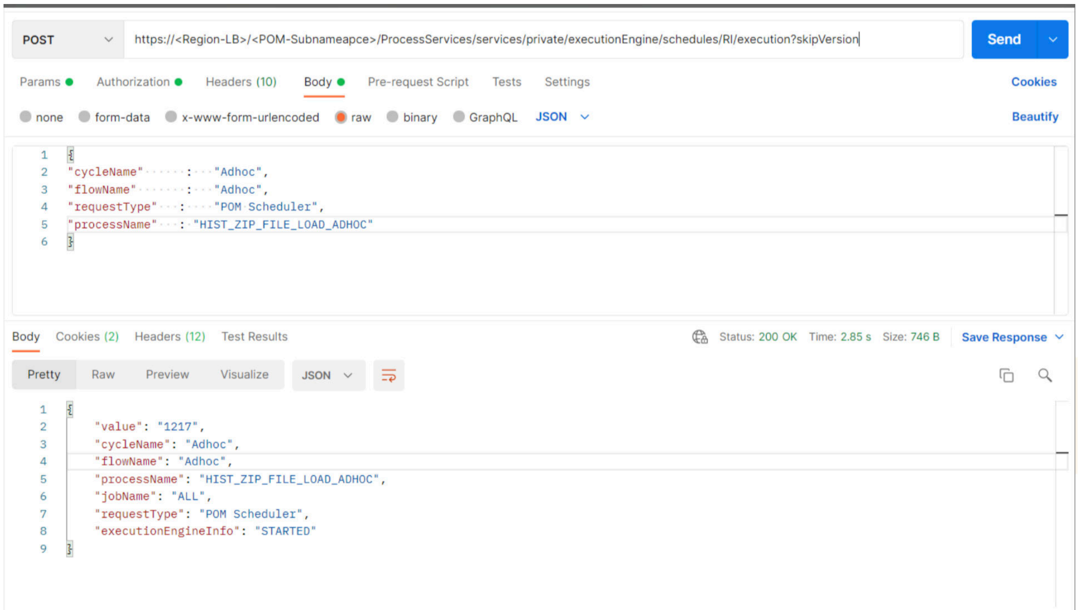
7. In the **Parameters**, provide a key of `skipVersion` with a value of `true`.



8. Click the **Body** tab to enter the JSON for running POM batches.
9. Select the **raw** radio button and **JSON** file type.
10. Enter the JSON XML and click the **Send** button in the **Body** tab.
    a. If the returned Status is "200 OK" and `'executionEngineInfo': "STARTED"` is in the response body, then the batch started as expected.

b. If the status is not 200 and either of 401 (authorization error) or 500 (incorrect body content) or 404 (server down) and so on, then perform error resolution as needed.

Example request and response for the `HIST_ZIP_FILE_LOAD_ADHOC` process:



Additional examples of Postman Body JSON XML are listed below.

Nightly Batch

```
{
"cycleName"    :  "Nightly",
"flowName"     :  "Nightly",
"requestType"  :  "POM Scheduler"
}
```

With One Process – Applicable for Nightly and Ad Hoc

```
{
"cycleName"     :  "Nightly",
"flowName"      :  "Nightly",
"requestType"   :  "POM Scheduler",
"processName"   :  "LOAD_AGGREGATION_BCOST_IT_DY_A_PROCESS"
}
```

With Dynamic Parameters – Applicable for Nightly and Ad Hoc

```
{
"cycleName"          :  "Adhoc",
"flowName"           :  "Adhoc",
"requestType"        :  "POM Scheduler",
"processName"        :  "HIST_INVRTV_LOAD_ADHOC",
"requestParameters" :  "jobParams.HIST_LOAD_INVRTV_DAY_JOB=2020-09-09
```

```
2021-09-09"
}
```

8

Data File Generation

When you are implementing the Retail Analytics and Planning without using an Oracle merchandising system for foundation data, or you are providing history data from non-Oracle sources, you will need to create several data files following the platform specifications. This chapter will provide guidance on the data file formats, structures, business rules, and other considerations that must be accounted for when generating the data.

> **Important:**
>
> Do not begin data file creation for RAP until you have reviewed this chapter and have an understanding of the key data structures used throughout the platform.

For complete column-level definitions of the interfaces, including datatype and length requirements, refer to the RI and AI Foundation Interfaces Guide in My Oracle Support. From the same document, you may also download Data Samples for all of the files covered in this chapter.

Files Types and Data Format

The shared platform data files discussed in this chapter may use a standard comma-delimited (CSV) file format, with text strings enclosed by quotation marks or other characters. The files must be UTF-8 encoded; other encoding types such as ANSI are not supported and may fail in the loads due to unrecognized characters. The files expect the first line to be column headers, and lines 2+ should contain the input data. For specific columns in each of these files, the following standards can be used as a guideline (though they can be changed by configuration options).

Table 8-1 Foundation File Formatting

Datatype	Format	Example	Explanation
Number	0.00	340911.10	Numbers should be unformatted with periods for decimal places. Commas or other symbols should not be used within the numerical values.
Character	“abc”	“Item #4561”	Any alphanumeric string can be optionally enclosed by quotation marks to encapsulate special characters such as commas in a descriptive value. Unique identifier columns, like `ITEM_ID`, that are `CHAR` or `VARCHAR2` datatypes require careful attention, because any characters in the string will be a part of that identifier, even spaces and underscores. For example, `" abc"` and `"abc "` are treated as different values unless you perform trimming on the field to remove spaces.

Table 8-1 (Cont.) Foundation File Formatting

Datatype	Format	Example	Explanation
Date	YYYYMMDD	20201231	Dates should be provided as simple 8-digit values with no formatting in year-month-day sequences.

Context Files

Before creating and processing a data file on the platform, choose the fields that will be populated and instruct the platform to only look for data in those columns. This configuration is handled through the use of Context (CTX) Files that are uploaded alongside each base data file. For example, the context file for `PRODUCT.csv` will be `PRODUCT.csv.ctx` (appending the `.ctx` file descriptor to the end of the base filename).

Within each context file you must provide a single column containing:

- One or more parameters defining the behavior of the file load and the format of the file.
- The list of fields contained in the source file, in the order in which they appear in the file specification:
  - #TABLE#<Staging Table Name>#
  - #DELIMITER#<Input Value>#
  - #DATEFORMAT#<Input Value>#
  - #REJECTLIMIT#<Input Value>#
  - #RECORDDELIMITER#<Input Value>#
  - #IGNOREBLANKLINES#<Input Value>#
  - #SKIPHEADERS#<Input Value>#
  - #TRIMSPACES#<Input Value>#
  - #TRUNCATECOL#<Input Value>#
  - #COLUMNLIST#<Input Value>#

    <COL1>

    <COL2>

    <COL3>

The following is an example context file for the `CALENDAR.csv` data file:

File Name: `CALENDAR.csv.ctx`

File Contents:

```
#TABLE#W_MCAL_PERIOD_DTS#
#DELIMITER#,#
#DATEFORMAT#YYYY-MM-DD#
#REJECTLIMIT#1#
#RECORDDELIMITER#\n#
#IGNOREBLANKLINES#false#
#SKIPHEADERS#1#
#TRIMSPACES#rtrim#
```

```
#TRUNCATECOL#false#
#COLUMNLIST#
MCAL_CAL_ID
MCAL_PERIOD_TYPE
MCAL_PERIOD_NAME
MCAL_PERIOD
MCAL_PERIOD_ST_DT
MCAL_PERIOD_END_DT
MCAL_QTR
MCAL_YEAR
MCAL_QTR_START_DT
MCAL_QTR_END_DT
MCAL_YEAR_START_DT
MCAL_YEAR_END_DT
```

The file must be UNIX formatted and have an end-of-line character on every line, including the last one. As shown above, the final EOL may appear as a new line in a text editor. The `#TABLE#` field is required: it indicates the name of the database staging table updated by the file. The `COLUMNLIST` tag is also required: it determines the columns the customer uses in their `.dat` or `.csv` file. The column list must match the order of fields in the file from left to right, which must also align with the published file specifications. Include the list of columns after the `#COLUMNLIST#` tag. Most of the other parameters are optional and the rows can be excluded from the context file. However, this will set values to system defaults that may not align with your format.

> **Note:**
>
> Both RI and AI Foundation can use these context files to determine the format of incoming data.

The server maintains a copy of all the context files used, so you do not need to send a context file every time. If no context files are found, the Analytics and Planning uses the last known configuration.

For additional format options, the available values used are from the DBMS_CLOUD package options in ADW.

If you want to retrieve the latest copy of the context files, the `RI_ZIP_UPLOAD_CTX_JOB` job in process `RI_ZIP_UPLOAD_CTX_ADHOC` can be run from the AIF DATA standalone schedule in POM. This job will extract all the context files from the `custom_ext_table_config` directory, package them in a zip file, and upload that file to Object Storage. The zip file is named `RAP_CTX.zip`, and will use `ris/outgoing` as the prefix for File Transfer Services (FTS) to access it.

In addition to being able to obtain copies of the files, there is also a database table named `C_DIS_ADW_EXT_TABLE_CONFIG` that holds the context file information that was last uploaded to the database. Except for the `COLUMN_LIST` and `FORMAT_OPTIONS` columns, the data in the other columns on the table is editable using the Control & Tactical Center screen in AI Foundation, so you can provide override values. The table does not have any initial data; it is populated when a CTX file is processed by `RI_UPDATE_TENANT_JOB`. When a CTX file is provided and data is also present in the table, the priority is to use the CTX file. If a CTX file was not provided in the current batch run, then the data on this table will be used. After the batch run, this table will reflect the most recently used CTX file configurations.

A change in format data in the table will trigger an update to ADW only if the values are different from what was last sent. This is done by comparing the entries at the `FORMAT_OPTIONS` column. Modifying the `COLUMN_LIST` in this table will not trigger a request to ADW to update the options. `COLUMN_LIST` is not editable through the Control & Tactical Center screen, as it only serves as a view to show the contents of the last payload sent to ADW. Sending the updates through a CTX file is the preferred method for modifying the column list. If no CTX files are provided, the `RI_UPDATE_TENANT_JOB` will end immediately instead of pushing the same configurations to ADW again. If you notice slow performance on this job then you can stop providing CTX files when they are not changing, and the job will finish within 10-20 seconds.

There is an `OVERRIDE_REJECTLIMIT_TO_DT` column on the table that will determine whether a `REJECTLIMIT` value other than zero is used. If this date column is null or is already past the current date, then the `REJECTLIMIT` will be reset to `0` and will trigger an update to ADW. The `REJECTLIMIT` value provided in the table will be used until the date specified in this column.

Application-Specific Data Formats

Each application within the Retail Analytics and Planning may require data to be provided using specific rules and data formats, which can differ from those used in the common platform files. This section describes the use-cases for alternate data formats and lays out the basic rules that must be followed.

Retail Insights

Retail Insights has a large number of legacy interfaces that do not follow the shared platform data formats. These interfaces are populated with files named after their target database table with a file extension of `.dat`, such as `W_PRODUCT_DS.dat`. All files ending with a `.dat` extension are pipe-delimited files by default (using the `|` symbol as the column separator) but can be changed using CTX file options. These files also have a Unix line-ending character by default, although the line-ending character can be configured to be a different value, if needed. These files may be created by a legacy Merchandising (RMS) extract process or may be produced through existing integrations to an older version of RI or AI Foundation.

Table 8-2 Retail Insights Legacy File Formatting

Datatype	Format	Example	Explanation
Number	0.00	340911.10	Unformatted numbers with periods for decimal places. Commas or other symbols cannot be used within the numerical values.
Character	abc	Item #4561	Any alphanumeric string will provided as-is, with the exception that it must NOT contain pipe characters or line-ending characters.
Date	YYYY-MM-DD;00:00:00	2020-05-09;00:00:00	Dates without timestamps must still use a timestamp format, but they must be hard-coded to have a time of 00:00:00. Date fields (such as `DAY_DT` columns) must NOT have a non-zero time, or they will not load correctly.
Timestamp	YYYY-MM-DD;HH:MM:SS	2020-05-09;09:35:19	Use full date-and-time formatting ONLY when a full timestamp is expected on the column. This is not commonly used and should be noted in the interface specifications, if supported.

This file format is used when integrating with legacy solutions such as the Retail Merchandising System (RMS) through the Retail Data Extractor (RDE) on v19 or earlier versions.

Example data from the file `W_RTL_PLAN1_PROD1_LC1_T1_FS.dat`:

```
70|-1|13|-1|2019-05-04;00:00:00|RETAIL|0|1118.82|1|70~13~2019-05-04;00:00:00~0
70|-1|13|-1|2019-05-11;00:00:00|RETAIL|0|476.09|1|70~13~2019-05-11;00:00:00~0
70|-1|13|-1|2019-05-18;00:00:00|RETAIL|0|296.62|1|70~13~2019-05-18;00:00:00~0
```

Retail AI Foundation Cloud Services

Modules within the AI Foundation Cloud Services leverage the same Context (CTX) file concepts as described in the common foundation file formats. You may control the structure and contents of AI Foundation files using the parameters in the context files. The full list of interfaces used by AI Foundation modules is included in the Interfaces Guide.

Planning Platform

Planning solutions using PDS (Planning Data Schema), such as Merchandise Financial Planning, have two main types of files:

Hierarchy/Dimension Files – Foundation Data for the Hierarchy/Dimensions.

Measure/Fact Files – Factual Data specific to loadable metric/measures.

When loading directly to Planning applications, both types of files should only be in CSV format and they should contain headers. Headers contain the details of the dimension names for Hierarchy/Dimension Files and the fact names for Measure/Fact Files.

Hierarchy/Dimension Files uses the naming convention `<Hierarchy Name>.hdr.csv.dat` and Measure Files can be any meaningful fact-grouping name, but with allowed extensions such as `.ovr`, `.rpl`, or `.inc`.

- OVR extension is used for override files
- RPL extension is used to delete and replace position-based data sets
- INC extension is for incremental files that can increment positional data.

If using the common foundation CSV files, most of the data can be interfaced using those shared integrations. However, certain files (such as the VAT Hierarchy) must be directly loaded to Planning: it does not come from the RAP data warehouse at this time. Refer to the application-specific Planning Implementation Guides for more details about the list of files that are not included in foundation integrations.

Dimension Files

A dimension is a collection of descriptive elements, attributes, or hierarchical structures that provide context to your business data. Dimensions tell the platform what your business looks like and how it operates. They describe the factual data (such as sales transactions) and provide means for aggregation and summarization throughout the platform. Dimensions follow a strict set of business rules and formatting requirements that must be followed when generating the files.

There are certain common rules that apply across all of the dimension files and must be followed without exception. Failure to adhere to these rules may result in failed data loads or incorrectly structured datasets in the platform.

- All dimension files must be provided as full snapshots of the source data at all times, unless you change the configuration of a specific dimension to be `IS_INCREMENTAL=Y` where incremental loads are supported. Incremental dimension loading should only be done once nightly/weekly batch processing has started. Initial/history dimension loads should always be full snapshots.
- Hierarchy levels must follow a strict tree structure, where each parent has a 1-to-N relationship with the children elements below them. You cannot have the same child level identifier repeat across more than one parent level, with the exception of Class/Subclass levels (which may repeat on the ID columns but must be unique on the UID columns). For example, Department 12 can only exist under Division 1, it cannot also exist under Division 2.
- Hierarchy files (product, organization, calendar) must have a value in all non-null fields for all rows and must fill in all the required hierarchy levels without exception. For example, even if your non-Oracle product data only has 4 hierarchy levels, you must provide the complete 7-level product hierarchy to the platform. Fill in the upper levels of the hierarchy with values to make up for the differences, such as having the division and group levels both be a single, hard-coded value.
- Any time you are providing a key identifier of an entity (such as a supplier ID, channel ID, brand ID, and so on) you should fill in the values on all rows of the data file, using a dummy value for rows that don't have that entity. For example, for items that don't have a brand, you can assign them to a generic "No Brand" value to support filtering and reporting on these records throughout the platform. You may find it easier to identify the "No Brand" group of products when working with CDTs in the AI Foundation Cloud Services or when creating dashboards in RI, compared to leaving the values empty in the file.
- Any hierarchy-level ID (department ID, region ID, and so on) or dimensional ID value (brand name, supplier ID, channel ID, store format ID, and so on) intended for Planning applications must not have spaces or special characters on any field, or it will be rejected by the PDS load. ID columns to be used in planning should use a combination of numbers, letters, and underscores only.
- Any change to hierarchy levels after the first dimension is loaded will be treated as a reclassification and will have certain internal processes and data changes triggered as a result. If possible, avoid loading hierarchy changes to levels above Item/Location during the historical load process. If you need to load new hierarchies during the history loads, make sure to advance the business date in the data warehouse using the specified jobs and date parameters, do NOT load altered hierarchies on top of the same business date as previous loads.
- All fields designated as flags (having FLG or FLAG in the field name) must have a `Y` or `N` value. Filters and analytics within the system will generally assume `Y`/`N` is used and not function properly if other values (like `0`/`1`) are provided.
- Retail Insights requires that all hierarchy identifiers above item/location level MUST be numerical. The reporting layer is designed around having numerical identifiers in hierarchies and no data will show in reports if that is not followed. If you are not implementing Retail Insights, then alphanumeric hierarchy IDs could be used, though it is not preferred.

Product File

The product file is named `PRODUCT.csv`, and it contains most of the identifying information about the merchandise you sell and the services you provide. The file structure follows certain rules based on the Retail Merchandising Foundation Cloud Services (RMFCS) data model, as that is the paradigm for retail foundation data that we are following across all RAP foundation files.

The columns below are the minimum required data elements, but the file supports many more optional fields, as listed in the Interfaces Guide. Optional fields tend to be used as reporting attributes in RI and are nullable descriptive fields. Optional fields designated for use in a AI Foundation or Planning module are generally nullable too, but should generally be populated with non-null values to provide more complete data to those modules.

Table 8-3 Product File Required Fields

Column Header	Usage
ITEM	Product number which uniquely identifies the record. Could be any of these records: • a sub-transaction level item (such as a UPC) which has a SKU as a parent and Style as a grandparent • a transaction-level SKU (with or without parents) or pack item • a style or "level 1" item which is above transaction level Style/colors are NOT considered as items and do not need to be provided as separate records.
ITEM_PARENT	Parent item associated with this record when the item level is 2 or 3. If you are not providing level 2 or 3 item records, then you may set this value to `-1` on all rows as no items will have parents. REQUIRED when providing multiple levels of items to establish the parent/child relationships.
ITEM_GRANDPARENT	Grandparent item associated with this record when the item level is 3. If you are not providing level 3 item records, then you may set this value to `-1` on all rows as no items will have grandparents. REQUIRED when providing multiple levels of items to establish the parent/child relationships.
ITEM_LEVEL	Item Level (`1`, `2`, or `3`) of this record from the source system. Used to determine what level of item is being provided to the target systems. Item level `2` should have a parent item, and item level `3` should provide both parent and grandparent. Typical fashion item levels are: • Level 1 – Style • Level 2 – SKU (transaction level) • Level 3 – UPC/EAN/barcode
TRAN_LEVEL	Transaction level (`1` or `2`) of the item from the source system. Identifies which level is used as the transaction level in RI and AI Foundation.
PACK_FLG	Pack flag (where `N` = regular item, `Y` = pack item). Defaults to `N` if not provided. REQUIRED column if the retailer has packs, optional otherwise. If pack items are going to be included, then also note that additional interfaces `SALES_PACK.csv` and `PROD_PACK.csv` become required.
DIFF_AGGREGATE	Combined differentiator values are used in defining the diff aggregate level (in between Item level `1` and `2` for a multi-level item). For example, for a fashion item, this will be the Color. Specify this on the transaction item-level records. This is used to dynamically create the planning item-parent (SKUP) level, so it must follow RPAS format rules (a combination of numbers, letters, and underscores only, no spaces or other characters).

Table 8-3 (Cont.) Product File Required Fields

Column Header	Usage
DIFF_AGGREGATE_DESC	Optional description associated with the `DIFF_AGGREGATE` ID provided above, to be used on the integration to PDS applications when constructing the planning item-parent level descriptions. If no description is provided, then the `DIFF_AGGREGATE` value will be used in both the IDs and descriptions.
LVL4_PRODCAT_ID	Default level for Subclass, which is the first level above item in the hierarchy structure. Sometimes referred to as segment or subsegment. All items of any type are mapped to a subclass as their first level. Parent items are not to be treated as hierarchy levels in the file.
LVL4_PRODCAT_UID	Unique identifier of the Subclass. In many merchandising systems the subclass is not unique on its own, so a separate, unique key value must be provided in this case.
LVL5_PRODCAT_ID	Default level for Class (sometimes referred to as Subcategory).
LVL5_PRODCAT_UID	Unique identifier of the Class. In many merchandising systems the class is not unique on its own, so a separate unique key value must be provided in this case.
LVL6_PRODCAT_ID	Default Level for Department (also referred to as Category).
LVL7_PRODCAT_ID	Default Level for Group.
LVL8_PRODCAT_ID	Default Level for Division.
TOP_PRODCAT_ID	Default Level for Company. Only one company is supported at this time, you may not have 2+ companies in the same dataset. Typically, this is hard-coded to a value of `1` for the company ID.
ITEM_DESC	Product Name or primary item description. When you are providing multiple levels of items, this may contain the style name, SKU name, or sub-transaction item name (for example, UPC description).
LVL4_PRODCAT_DESC	Default Level for Subclass Description.
LVL5_PRODCAT_DESC	Default Level for Class Description.
LVL6_PRODCAT_DESC	Default Level for Department Description.
LVL7_PRODCAT_DESC	Default Level for Group Description.
LVL8_PRODCAT_DESC	Default Level for Division Description.
TOP_PRODCAT_DESC	Default Level for Company Description.
INVENTORIED_FLG	Indicates whether the item carries stock on hand. Data sent to Planning apps is generally only for inventoried items (`Y`), but you may have non-inventoried items loaded for other purposes (`N`). The flags are used from the SKU level item records, values on the style or UPC level item records can be defaulted to some value, they are not currently used.
SELLABLE_FLG	Indicates whether the item is sellable to customers. Data sent to Planning apps is only for sellable items (`Y`) or pack items (`PACK_FLG=Y`), but you may have non-sellable items loaded for other purposes (`N`). The flags are used from the SKU level item records, values on the style or UPC level item records can be defaulted to some value; they are not currently used.

The product hierarchy fields use generic level names to support non-traditional hierarchy structures (for example, your first hierarchy level may not be called Subclass, but you are still loading it into the same position in the file). Other file columns such as `LVL1` to `LVL3` exist in the interface but are not yet used in any module of the platform.

> **Note:**
>
> Multi-level items are not always required and depend on your use-cases. For example, the lowest level (`ITEM_LEVEL=3`) for sub-transaction items is only used in Retail Insights for reporting on UPC or barcode level attribute values. Most implementations will only have `ITEM_LEVEL=1` and `ITEM_LEVEL=2` records. If you are a non-fashion retailer you may only have a single item level (for SKUs) and the other levels could be ignored. The reason for having different records for each item level is to allow for different attributes at each level, which can be very important in Retail Insights analytics. You may also need to provide multiple item levels for optimizing or planning data at a Style or Style/Color level in the non-RI modules. When providing multiple item level records, note that the item IDs must be unique across all levels and records.

Example data for the `PRODUCT.csv` file columns above, including all 3 supported item levels (style, SKU, and UPC):

```
ITEM,ITEM_PARENT,ITEM_GRANDPARENT,ITEM_LEVEL,TRAN_LEVEL,PACK_FLG,DIFF_AGGREGAT
E,LVL4_PRODCAT_ID,LVL4_PRODCAT_UID,LVL5_PRODCAT_ID,LVL5_PRODCAT_UID,LVL6_PRODC
AT_ID,LVL7_PRODCAT_ID,LVL8_PRODCAT_ID,TOP_LVL_PRODCAT_ID,ITEM_DESC,LVL4_PRODCA
T_DESC,LVL5_PRODCAT_DESC,LVL6_PRODCAT_DESC,LVL7_PRODCAT_DESC,LVL8_PRODCAT_DESC
,TOP_LVL_PRODCAT_DESC,INVENTORIED_FLG,SELLABLE_FLG
190085210200,-1,-1,1,2,N,,8,9001,3,910,3,2,1,1,2IN1 SHORTS,Shorts,Active
Apparel,Women's Activewear,Activewear,Apparel,Retailer Ltd,Y,Y
190085205725,190085210200,-1,2,2,N,BLK,8,9001,3,910,3,2,1,1,2IN1
SHORTS:BLACK:LARGE,Shorts,Active Apparel,Women's
Activewear,Activewear,Apparel,Retailer Ltd,Y,Y
190085205923,190085210200,-1,2,2,N,DG,8,9001,3,910,3,2,1,1,2IN1 SHORTS:DARK
GREY:LARGE,Shorts,Active Apparel,Women's
Activewear,Activewear,Apparel,Retailer Ltd,Y,Y
1190085205725,190085205725,190085210200,3,2,N,,8,9001,3,910,3,2,1,1,2IN1
SHORTS:BLACK:LARGE:BC,Shorts,Active Apparel,Women's
Activewear,Activewear,Apparel,Retailer Ltd,Y,Y
1190085205923,190085205923,190085210200,3,2,N,,8,9001,3,910,3,2,1,1,2IN1
SHORTS:DARK GREY:LARGE:BC,Shorts,Active Apparel,Women's
Activewear,Activewear,Apparel,Retailer Ltd,Y,Y
```

This example and the field descriptions covered in this section all follow the standard Merchandising Foundation (RMFCS) structure for product data, and it is strongly recommended that you use this format for RAP. If you are a legacy Planning customer or have specific needs for extended hierarchies, the preferred approach is to convert your non-RMS hierarchy structure to a standard RMS-like foundation format. This conversion involves:

- Provide only the SKUs and Styles as separate item records (dropping the style/color level from the hierarchy). The Style will be the `ITEM_PARENT` value on the SKU records and `ITEM_GRANDPARENT` will always be `-1`.
- Populate the field `DIFF_AGGREGATE` at the SKU level with the differentiator previously used in the style/color level. For example, a legacy style/color item ID of `S1000358:BLUE` will

instead create S1000358 as the ITEM for the style-level record and the ITEM_PARENT in the SKU record. The value BLUE is written in the DIFF_AGGREGATE field in the SKU-level record (DIFF_AGGREGATE can be set to -1 or left null on style level records).

- When constructing the extended hierarchies in Planning and AI Foundation, the styles and diff aggregate values are concatenated together to dynamically create the style/color level of the hierarchy where needed.

Following this approach for your product hierarchy ensures you are aligned with the majority of Oracle Retail applications and will be able to take up additional retail applications in the future without restructuring your product data again.

For other fields not shown here, they are optional from a data load perspective but may be used by one or more applications on the platform, so it is best to consider all fields on the interface and populate as much data as you can. For example, supplier information is a requirement for Inventory Planning Optimization, and brand information is often used in Clustering or Demand Transference. Also note that some fields come in pairs and must be provided together or not at all. This includes:

- Brand name and description
- Supplier ID and description

Description fields can be set to the same value as the identifier if no other value is known or used, but you must include both fields with non-null values when you want to provide the data.

Product Alternates

You may also use the file PRODUCT_ALT.csv to load additional attributes and hierarchy levels specifically for use in Planning applications. The file data is always at item level and may have up to 30 flexible fields for data. These columns exist in the PRODUCT.csv file if you are a non-RMFCS customer so this separate file would be redundant. If you are using RMFCS, then this file provides a way to send extra data to Planning that does not exist in RMFCS.

When using flex fields as alternate hierarchy levels, there are some rules you will need to follow:

- All hierarchies added this way must have an ID and Description pair as two separate columns
- The ID column for an alternate hierarchy must ONLY contain numbers; no other characters are permitted

Numerical ID fields are required for integration purposes. When a plan is generated in MFP or AP using an alternate hierarchy, and you wish to send that plan data to AIF for in-season forecasting, the alternate hierarchy ID used must be a number for the integration to work. If your alternate hierarchy level will not be used as the base intersection of a plan, then it does not need to be limited to numerical IDs (although it is still recommended to do so). This requirement is the same for all hierarchy levels when Retail Insights is used, as RI can only accept numerical hierarchy IDs for all levels (for both base levels and alternates).

For example, you might populate FLEX1_CHAR_VALUE with numerical IDs for an alternate level named “Subsegment”. You will put the descriptions into FLEX2_CHAR_VALUE. These values can be mapped into PDS by altering the interface.cfg file, and the values may be used to define plans or targets in MFP. When you export your plans for AIF, they are written into integration tables such as MFP_PLAN1_EXP using the numerical identifiers from FLEX1_CHAR_VALUE as the plan level. This is further integrated to RI tables like W_RTL_PLAN1_PROD1_LC1_T1_FS (columns ORG_DH_NUM and PROD_DH_NUM for location/product IDs respectively). This is where numerical IDs become required for these interfaces to function; they will not load the data if the IDs are non-numerical. Once loaded into `W_RTL_PLAN1_PROD1_LC1_T1_F` and similar tables, AIF reads the plan data to feed in-season forecast generation.

Loading the data into data warehouse tables at a flex field level requires additional configuration. Refer to the *RI Implementation Guide* for details. AIF also requires additional setup to use alternate hierarchies. Refer to the section “Building Alternate Hierarchy in AIF” in the *AIF Implementation Guide* for details.

Re-Using Product Identifiers

It may happen over time that the same product keys (such as SKU numbers) will be re-used to represent brand new items. There are two parameters in `C_ODI_PARAM_VW` that must be updated to enable this functionality (`RI_ITEM_REUSE_IND` and `RI_ITEM_REUSE_AFTER_DAYS`). Set `RI_ITEM_REUSE_IND` to `Y` and set `RI_ITEM_REUSE_AFTER_DAYS` to some number greater than 0. The combination of these parameters will alter the product hierarchy load in the following ways:

1. Once enabled, if an item is deleted (when using incremental loads) or stops appearing in the nightly product files (when using full snapshot loads) it will NOT be closed, it will remain as `current_flg=Y` for the number of days specified in the configuration.
2. If the item re-appears in the product file the next night, then the existing item record remains open with `current_flg=Y` and it will be as if it was never dropped or deleted.
3. If the item re-appears in the product file only after the set number of days has elapsed, then the old version of the product is both closed and archived (example below). The incoming item record is inserted as a new item with no history.

Here is an example of re-using an item:

1. Item number 1001 is created as a new item in a source system, such as RMFCS.
2. Item exists for a period of time while accumulating fact data such as sales and inventory.
3. Item becomes inactive and is eventually deleted from the source system, which marks it inactive in RAP after the set number of days for re-use has passed.
4. After another period of time (such as 1 month) the same item number 1001 is added back in the source system, representing a completely new item with different attributes and hierarchies. The window between the old and new item is configured in `C_ODI_PARAM_VW` as mentioned in the Setup and Configuration chapter.
5. When item 1001 comes to RAP again, the old version of this item will be archived using a value appended to the code (for example, 1001_230101) across all product tables in the data warehouse (the item data would already have been dropped from Planning when the re-use number of days had elapsed). Item 1001 is then inserted as a new item not associated in any way with the prior instance of that ID. The history that already exists for item 1001 is now associated with the archived item 1001_230101, and new fact data will be associated only with the new definition of item 1001 going forward.

The same process flow applies both when you are creating the `PRODUCT.csv` file from a non-Oracle source and when you use RMFCS to provide the data.

Organization File

The organization file will contain most of the identifying information about the locations where you sell or store merchandise, including physical locations (such as a brick & mortar store) and virtual locations (such as a web store or virtual warehouse entity). The file structure follows certain rules based on the Retail Merchandising Foundation Cloud Services (RMFCS) data model, as that is the paradigm for retail foundation data that we are following across all RAP foundation files. The columns below are the minimum required data elements, but the file supports many more optional fields, as listed in the Interfaces Guide.

Table 8-4 Organization File Required Fields

Column Header	Usage
ORG_NUM	The external identifier for a location, including stores, warehouses, and partner locations. This value MUST be a number if you will use Retail Insights. RI cannot use non-numeric organization IDs.
ORG_TYPE_CODE	The type code of the location. It must be one of `S`, `W`, or `E`, representing a Store, Warehouse, or External location.
CURR_CODE	This is the 3-character base currency code of the organizational entity, such as `USD` or `CAN`.
ORG_HIER10_NUM	Default Level for District, which is the first level above Location in the hierarchy. Hierarchy values MUST be a number if you will use Retail Insights. RI cannot use non-numeric hierarchy IDs.
ORG_HIER11_NUM	Default Level for Region.
ORG_HIER12_NUM	Default Level for Area. Also referred to as the Channel level in some Planning applications.
ORG_HIER13_NUM	Default Level for Chain.
ORG_TOP_NUM	Default Level for Company. Only one company is supported at this time. You may not have 2+ companies in the same instance.
ORG_DESC	Short name or short description of the location.
ORG_SECONDARY_DESC	Full name or long description of the location.
ORG_HIER10_DESC	Default Level for District Description.
ORG_HIER11_DESC	Default Level for Region Description.
ORG_HIER12_DESC	Default Level for Area Description. Also referred to as the Channel level in some Planning applications.
ORG_HIER13_DESC	Default Level for Chain Description.
ORG_TOP_DESC	Default Level for Company Description.
PHYS_WH_ID	The physical warehouse ID linked to a virtual warehouse. Must be specified for warehouses, can be null otherwise. A physical warehouse record can have its own ID as the value here. A virtual warehouse should have the linked physical warehouse ID that contains the virtual location.
VIRTUAL_WH_FLG	Indicates whether the warehouse record is a physical or virtual WH. Planning GA solutions only use virtual WHs so flag must be `Y` to send the WH to Planning. Physical warehouse records (`VIRTUAL_WH_FLG=N`) are not used by planning applications but are used for AI Foundation (such as for Inventory Planning Optimization).

The organization hierarchy fields use generic level names to support non-traditional hierarchy levels (for example, your first hierarchy level may not be called District, but you are still loading it into the same position in the file which is used for Districts). Other levels, such as `1` to `9`, have columns in the interface but are not yet used in any module of the platform.

Warehouses get special handling both in the input interface load and throughout the RAP applications. Warehouses are not considered a part of the organization hierarchy structure. While you are required to put some value in the hierarchy level fields for warehouses (because the columns are not nullable) those values are not currently used. Instead, the values will be discarded and the warehouses are loaded with no parent levels in the data warehouse tables. You should provide a unique reserved value like `1` or `9999` on all hierarchy level numbers between location and company for warehouses, just to ensure the data is loaded without violating any multi-parentage rules. When exporting the warehouse locations to Planning applications, each warehouse ID is assigned its own name and number for each parent level, prefixed with `WH` to make the level IDs distinct from any store hierarchy level. The warehouses must then be mapped to channels from the MFP user interface before you can use their data.

Example data for the `ORGANIZATION.csv` file columns above as well as some optional fields available on the interface:

```
ORG_NUM,ORG_TYPE_CODE,CURR_CODE,STATE_PROV_NAME,COUNTRY_REGION_NAME,ORG_HIER10
_NUM,ORG_HIER11_NUM,ORG_HIER12_NUM,ORG_HIER13_NUM,ORG_TOP_NUM,ORG_DESC,ORG_SEC
ONDARY_DESC,ORG_HIER10_DESC,ORG_HIER11_DESC,ORG_HIER12_DESC,ORG_HIER13_DESC,OR
G_TOP_DESC,CHANNEL_ID,CHANNEL_NAME,PHYS_WH_ID,STOCKHOLDING_FLG,STORE_FORMAT_DE
SC,STORE_FORMAT_ID,STORE_TYPE,TRANSFER_ZONE_ID,TRANSFER_ZONE_DESC,VIRTUAL_WH_F
LG,STORE_CLASS_TYPE,STORE_CLASS_DESC,WH_DELIVERY_POLICY,WH_REPL_IND,DUNS_NUMBE
R,STORE_REMODEL_DT,STORE_CLOSE_DT,INBOUND_HANDLING_DAYS,FLEX1_CHAR_VALUE,FLEX2
_CHAR_VALUE,FLEX3_CHAR_VALUE,FLEX4_CHAR_VALUE,FLEX5_CHAR_VALUE,FLEX6_CHAR_VALU
E,FLEX7_CHAR_VALUE,FLEX8_CHAR_VALUE,FLEX9_CHAR_VALUE,FLEX10_CHAR_VALUE
1000,S,USD,North Carolina,United
States,1070,170,1,1,1,Charlotte,Charlotte,North Carolina,Mid-Atlantic,Brick &
Mortar,US,Retailer Ltd,1,North America,,Y,Store,1,C,101,Zone
101,N,1,A,,,,,,,WH-1,Warehouse - US,1,Store Pick Up / Take
With,3,Comp,6,Mixed Humid,1,Very Large
1001,S,USD,Georgia,United States,1023,400,1,1,1,Atlanta,Atlanta,Georgia,South
Atlantic,Brick & Mortar,US,Retailer Ltd,1,North America,,Y,Kiosk,2,C,101,Zone
101,N,6,F,,,,,,,WH-1,Warehouse - US,2,Deliver/Install at
Customer ,3,Comp,7,Hot Humid,3,Medium
1002,S,USD,Texas,United States,1104,230,1,1,1,Dallas,Dallas,Texas,Gulf
States,Brick & Mortar,US,Retailer Ltd,1,North America,,Y,Store,1,C,101,Zone
101,N,6,F,,,,,,,WH-1,Warehouse - US,3,Home Delivery,3,Comp,4,Hot Dry,3,Medium
```

It is important that your organization hierarchy follow the standard rules laid out at the beginning of this chapter. All IDs must be unique (within their level) and IDs can never be re-used under multiple parents. All IDs must be numbers if you are using Retail Insights. The entire 6-level structure must be filled out, even if your source system doesn't have that many levels.

> **Note:**
>
> You may duplicate a higher level down to lower levels if you need to fill it out to meet the data requirements.

Also note that some optional fields come in pairs and must be provided together or not at all. This includes:

- Banner ID and description
- Channel ID and description
- Store format ID and description

Description fields can be set to the same value as the identifier if no other value is known or used, but you must include both fields with non-null values when you provide the data.

Organization Alternates

You may also use the file `ORGANIZATION_ALT.csv` to load additional attributes and hierarchy levels specifically for use in Planning applications. The file data is always at location level and may have up to 30 flexible fields for data. These columns exist on the `ORGANIZATION.csv` file if you are a non-RMFCS customer, so this separate file would be redundant. If you are using RMFCS, then this file provides a way to send extra data to Planning that does not exist in RMFCS.

When using flex fields as alternate hierarchy levels, there are some rules you will need to follow:

- All hierarchies added this way must have an ID and Description pair as two separate columns
- The ID column for an alternate hierarchy must ONLY contain numbers, no other characters are permitted

Numerical ID fields are required for integration purposes. When a plan is generated in MFP or AP using an alternate hierarchy, and you wish to send that plan data to AIF for in-season forecasting, the alternate hierarchy ID used must be a number for the integration to work. If your alternate hierarchy level will not be used as the base intersection of a plan, then it does not need to be limited to numerical IDs (although it is still recommended to do so). This requirement is the same for all hierarchy levels when Retail Insights is used, as RI can only accept numerical hierarchy IDs for all levels (both base levels and alternates).

For example, you might populate `FLEX1_CHAR_VALUE` with numerical IDs for an alternate level named “Subsegment”. You will put the descriptions into `FLEX2_CHAR_VALUE`. These values can be mapped into PDS by altering the `interface.cfg` file, and the values can be used to define plans or targets in MFP. When you export your plans for AIF, they are written into integration tables such as `MFP_PLAN1_EXP` using the numerical identifiers from `FLEX1_CHAR_VALUE` as the plan level. This is further integrated to RI tables like `W_RTL_PLAN1_PROD1_LC1_T1_FS` (columns `ORG_DH_NUM` and `PROD_DH_NUM` for location/product IDs respectively). This is where numerical IDs become required for these interfaces to function; they will not load the data if the IDs are non-numerical. Once loaded into `W_RTL_PLAN1_PROD1_LC1_T1_F` and similar tables, AIF reads the plan data to feed in-season forecast generation.

Loading the data into data warehouse tables at a flex field level requires additional configuration. Refer to the *RI Implementation Guide* for details. AIF also requires additional setup to use alternate hierarchies. Refer to the section “Building Alternate Hierarchy in AIF” in the *AIF Implementation Guide* for details.

Calendar File

The calendar file contains your primary business or fiscal calendar, defined at the fiscal-period level of detail. The most common fiscal calendar used is a 4-5-4 National Retail Federation (NRF) calendar or a variation of it with different year-ending dates. This calendar defines the financial, analytical, or planning periods used by the business. It must contain some form of fiscal calendar, but if you are a business that operates solely on the Gregorian calendar, a default calendar file can be generated by an ad hoc batch program to initialize the system. However, if you are implementing a planning solution, you must use the Fiscal Calendar as your primary calendar, and only this calendar will be integrated from the data warehouse to Planning.

Table 8-5 Calendar File Required Fields

Column Header	Usage
MCAL_CAL_ID	Identifies the accounting calendar. At this time a hard-coded value of `Retail Calendar~41` is expected here, no other value should be used.
MCAL_PERIOD_TYPE	Identifies the accounting period type (`4` or `5`). This represents if the fiscal period has 4 or 5 weeks.
MCAL_PERIOD_NAME	Name of the fiscal period, such as `Period01`, `Period02`, and so on. Do not include the year in this name, as the system will automatically add that during the load.
MCAL_PERIOD	Period number within a year, for a 4-5-4 calendar this should be between `1` and `12`.
MCAL_PERIOD_ST_DT	Identifies the first date of the period in YYYYMMDD format (default format can be changed in CTX files).
MCAL_PERIOD_END_DT	Identifies the last date of the period in YYYYMMDD format (default format can be changed in CTX files).
MCAL_QTR	Identifies the quarter of the year to which this period belongs. Possible values are `1`, `2`, `3` and `4`.
MCAL_YEAR	Identifies the fiscal year in YYYY format.
MCAL_QTR_START_DT	Identifies the start date of the quarter in YYYYMMDD format (default format can be changed in CTX files).
MCAL_QTR_END_DT	Identifies the end date of the quarter in YYYYMMDD format (default format can be changed in CTX files).
MCAL_YEAR_START_DT	Identifies the start date of the year in YYYYMMDD format (default format can be changed in CTX files).
MCAL_YEAR_END_DT	Identifies the end date of the year in YYYYMMDD format (default format can be changed in CTX files).

The hard-coded calendar ID is used to align with several internal tables that are designed to support multiple calendars but currently have only one in place, and that calendar uses the provided value of `MCAL_CAL_ID` above.

The fiscal calendar should have, at a minimum, a 5-year range (2 years in the past, the current fiscal year, and 2 years forward from that) but is usually much longer so that you do not need to update the file often. Most implementations should start with a 10-15 year fiscal calendar length. The calendar should start at least 1 full year before the planned beginning of your history files and extend at least 1 year beyond your expected business needs in all RAP modules.

Example data for the `CALENDAR.csv` file columns above:

```
MCAL_CAL_ID,MCAL_PERIOD_TYPE,MCAL_PERIOD_NAME,MCAL_PERIOD,MCAL_PERIOD_ST_DT,MC
AL_PERIOD_END_DT,MCAL_QTR,MCAL_YEAR,MCAL_QTR_START_DT,MCAL_QTR_END_DT,MCAL_YEA
R_START_DT,MCAL_YEAR_END_DT
Retail
Calendar~41,4,Period01,1,20070204,20070303,1,2007,20070204,20070505,20070204,2
0080202
Retail
Calendar~41,5,Period02,2,20070304,20070407,1,2007,20070204,20070505,20070204,2
0080202
```

```
Retail
Calendar~41,4,Period03,3,20070408,20070505,1,2007,20070204,20070505,20070204,2
0080202
Retail
Calendar~41,4,Period04,4,20070506,20070602,2,2007,20070506,20070804,20070204,2
0080202
```

Exchange Rates File

The exchange rates file captures conversion rates between any two currency codes that may appear in your fact data. The standard practice for fact data is to load the source system values in the original currency and allow the platform to convert the amounts to the primary currency. This file facilitates that process and triggers bulk updates in nightly processing any time you wish to change your exchange rates for financial reporting purposes. Adding new rates to the file with an effective date equal to the batch date triggers a mass update to all positional facts, converting all the amounts to the new exchange rate even if the item/location did not otherwise change in the source system.

Note that you do not have to provide exchange rates data for the following scenarios:

- You are loading your data already converted to primary currency
- You only use a single currency for your entire business
- You are only implementing a AI Foundation module and expect to perform those processes in the local currency amounts

Even when exchange rates are not required as a separate file, you must still populate the currency codes (`DOC_CURR_CODE`, `LOC_CURR_CODE`) in the fact data files with values. Review the scenarios below to understand how to set these values and provide the associated rates.

Scenario 1 - No Conversion

For this use-case, all data is in the desired currency before sending it to Oracle. You do not want the platform to convert your data from source currency to primary currency. All fact records must have `LOC_CURR_CODE` = `DOC_CURR_CODE`. For example, set both values to `USD` for sales in the U.S. and both values to `CAD` for sales in Canada that you pre-converted. `EXCH_RATE.csv` data is not required or used for records having the same currency code on both columns.

Scenario 2 – Only One Currency

If your business only operates in one region and uses a single currency code, then you don't need to provide exchange rate data. All fact records must have `LOC_CURR_CODE` = `DOC_CURR_CODE`. For example, set both values to `USD` on all rows if your primary operating currency is USD. `EXCH_RATE.csv` data is not required or used for records having the same currency code on both columns.

Scenario 3 – Multiple Currencies

When you do plan to provide data in multiple source currencies, you must also provide the exchange rates into and out of those currencies. Your fact data must have `DOC_CURR_CODE` set to the currency of the source system where the transaction was recorded (for example, a sale in Canada has a document currency of `CAD`). The value of `LOC_CURR_CODE` will be the same on all records and must be the primary operating currency of your business (if you operate mainly in the United States then it will be `USD`).

Exchange rates should be provided using the standard international rates (for example USD > CAD may be 1.38) but the fact load will perform lookups in reverse. Fact conversions are applied as a division process. For example, “transaction amount / exchange rate” is the formula to convert from document currency to primary currency; so when converting from CAD > USD the system will look up the value for USD > CAD and divide by that number to get the final value.

Table 8-6 Exchange Rate File Required Fields

Column Header	Usage
START_DT	Contains the effective start date of the exchange rate. Set to the current business date to trigger new rate conversions.
END_DT	Contains the effective end date of the exchange rate. Default to `21000101` if the rate should be effective indefinitely.
EXCHANGE_RATE	Contains the exchange rate for the specified currency/type/ effective date combination.
FROM_CURRENCY_CODE	Code of the currency to be exchanged.
TO_CURRENCY_CODE	Code of the currency to which a currency is exchanged.

Sample data for the EXCH_RATE.csv file columns above:

```
START_DT,END_DT,EXCHANGE_RATE,FROM_CURRENCY_CODE,TO_CURRENCY_CODE
20180514,21000101,0.8640055,CAD,USD
20180514,21000101,0.1233959,CNY,USD
```

The exchange rates data must also satisfy the following criteria if you are loading data for use in Retail Insights reporting:

1. Rates must be provided in both directions for every combination of currencies that can occur in your dataset (for example, USD > CAD and CAD > USD).
2. Dates must provide complete coverage of your entire timeframe in the dataset, both for historical and current data. The current effective records for all rates can use 2100-01-01 as the end date. Dates cannot overlap, only a single rate must be effective per day.
3. Rates should not change more often than absolutely necessary based on the business requirements. If you are implementing RI with positional data, a rate change triggers a complete recalculation of the stock on hand cost/retail amounts for the entire business across all pre-calculated aggregate tables. When RI is not used for financial reporting you might only change the rates once each fiscal year, to maintain a single constant currency for analytical purposes.

Attributes Files

Product attributes are provided on two files: one file for the attribute-to-product mappings and another for attribute descriptions and codes. These files should be provided together to fully describe all the attributes being loaded into the system. The attribute descriptors file must be a full snapshot of all attribute types and values at all times. The product attribute mapping file should start as a full snapshot but can move to incremental (delta) load methods once nightly batches begin, if you can extract the information as deltas only.

Product attributes are a major component of the RI and AI Foundation modules and drive many analytical processes but are not required for some planning modules like MFP.

Table 8-7 Attribute File Required Fields

Column Header	Usage
ATTR_VALUE_ID	Unique identifier for a user-defined attribute or product differentiator. This interface contains ALL values regardless of whether they are used on items yet or not. These values must also match the `ATTR_ID` on the other file.
ATTR_VALUE_DESC	Descriptive value for a user-defined attribute or product differentiator, such as `Brown`, `Cotton`, `Size12`, and so on
ATTR_GROUP_ID	Unique identifier for a group of user-defined attributes, or the name/ code for the differentiator group. Reserved attribute types like `SIZE` and `COLOR` must all have a single, hard-coded value associated with the group in this field (For example, all sizes must be in the `SIZE` group, don't specify group IDs for `sizes_pants`, `sizes_shirts`, and so on. This is handled separately). This group ID value must also match the `ATTR_GRP_ID` value on the second file.
ATTR_GROUP_DESC	Descriptive value for a group of user-defined attributes, such as `Color Family` or `Web Exclusive Code`, or the description of the differentiator type such as `Color`.
ATTR_TYPE_CODE	Indicates the type of UDA or differentiator. UDA types should be hard-coded as one of `FF`, `LV`, or `DT`, for Freeform, List of Values, or Date. `LV` type attributes are fixed lists and write values to translation lookup tables, while `FF` and `DT` fields do not. For non-UDA types, some diffs have special reserved codes for this field as well, which should be used when applicable, and include `SIZE`, `FABRIC`, `SCENT`, `FLAVOR`, `STYLE`, and `COLOR`. Other differentiators not using these codes should specify a hard-coded type of `DIFF`.

Sample data for the `ATTR.csv` file columns above:

```
ATTR_VALUE_ID,ATTR_VALUE_DESC,ATTR_GROUP_ID,ATTR_GROUP_DESC,ATTR_TYPE_CODE
13,No_Sugar_IN13,45008,UDA_ING_2018.01.16.01.00,FF
14,Zero_Carbs_IN14,45008,UDA_ING_2018.01.16.01.00,FF
3,Distressed,80008,Wash,LV
STEEL,Steel,METAL,Metals,DIFF
CHOC,Chocolate,FLAVOR,Flavor,FLAVOR
GRAY_43214,Gray,COLOR,Color,COLOR
32X32_9957,32X32,SIZE,Size,SIZE
```

Table 8-8 Product Attribute File Required Fields

Column Header	Usage
ITEM	The item number associated with the specified attribute value.
ATTR_ID	Identifier for an attribute value, such as the UDA value ID or Diff ID which is mapped to the item on the record, or the ID for the item list this item belongs to. These values must also match the `ATTR_VALUE_ID` on the other file.

Table 8-8 (Cont.) Product Attribute File Required Fields

Column Header	Usage
ATTR_GRP_TYPE	The attribute group type. This is a set of fixed values which must be selected from what RAP supports. Supported values are `ITEMDIFF`, `ITEMUDA`, `ITEMLIST`, `COLOR`, and `PRODUCT_ATTRIBUTES`. These codes determine the target columns for the data (for example, lists, diffs, and UDAs use different internal columns in the data model). `PRODUCT_ATTRIBUTES` type code encapsulates the other named differentiator types like Size, Fabric, and so on. `COLOR` has a special type code due to it being a common level between Style and SKU for fashion retailers, so it is handled separately.
ATTR_GRP_ID	Identifier for the attribute group containing the value on this record. Must match a `ATTR_GROUP_ID` in the other file. Varies by `ATTR_GRP_TYPE` value used: • If `ATTR_GRP_TYPE` is in `ITEMDIFF`, `COLOR`, `PRODUCT_ATTRIBUTES`, then specify the Diff Type ID or one of the reserved values like `SIZE` or `COLOR`. • If `ATTR_GRP_TYPE` is `ITEMUDA`, specify UDA Group ID. • If `ATTR_GRP_TYPE` is `ITEMLIST`, this field is not used, leave null.
DIFF_GRP_ID	Differentiator group used to assign the diff attribute on this item; for example the Size Group ID used when generating SKUs from a parent Style. Only SKUs will have diff groups associated with them. Only diffs will have groups, not UDAs or other record types. This is not the same as an attribute group, which is the overall grouping of attribute values across all items. This is used mainly for Size Profile Science. Foreign key reference to `DIFF_GROUP.csv`.
DIFF_GRP_DESC	Descriptive value of the diff group mapped to this item/attribute record.

Sample data for the `PROD_ATTR.csv` file columns above:

```
ITEM,ATTR_ID,ATTR_GRP_TYPE,ATTR_GRP_ID,DIFF_GRP_ID,DIFF_GRP_DESC
91203747,13,ITEMUDA,45008,,
91203747,3,ITEMUDA,80008,,
190496585706,STEEL,ITEMDIFF,METAL,,
86323133004,GRAY_43214,COLOR,COLOR,,
190085302141,CHOC,PRODUCT_ATTRIBUTES,FLAVOR,,
345873291,32X32_9957,PRODUCT_ATTRIBUTES,SIZE,S13,Pant Sizes
```

Fact Files

Fact Data Key Columns

Fact data files, such as sales and inventory, share many common characteristics and standards that can be followed regardless of the specific file. The table below summarizes those key elements and their minimum requirements. You will generally always want to populate these fields where they exist on an interface file.

Table 8-9 Common Fact Data Fields

Column Header	Usage
ITEM	Unique identifier of a transaction item. Must align with the product records in the PRODUCT.csv file where ITEM_LEVEL = TRAN_LEVEL.
ORG_NUM	Unique identifier of an organizational entity. Must align with the location records in the ORGANIZATION.csv file.
DAY_DT	Transaction date or business date for the fact data entry, formatted as YYYYMMDD. Must be a valid date within the range of periods in CALENDER.csv.
RTL_TYPE_CODE	Retail types define a general category for the record that varies by interface. For Sales and Markdowns, it must be one of R/P/C, representing the regular/promo/clearance status of the transaction.
*_QTY	Fields ending with QTY represent the quantity or units of an item on the record, such as the units sold on a transaction line or the units of on-hand inventory.
*_AMT_LCL	Fields containing AMT_LCL represent the currency amount of a record in the local currency of the source system, such as sales retail amount in Canadian dollars from a Canada location, or the cost value of on-hand inventory at your U.S. warehouse.
DOC_CURR_CODE	The original document currency of the record in the source system. For example, a sale made in Canada may have a value of CAD in this field.
LOC_CURR_CODE	The local operating currency of your main office or headquarters. A company based in the United States would use USD in this field.
ETL_THREAD_VAL	If you are providing any data files ending in a .dat extension, then it might contain an ETL_THREAD_VAL column. This column must be hard-coded to be 1 on all rows without exception; it should not be null and should not be any values greater than 1. This is a legacy field that allowed multi-threading in older generations of RAP architecture.

Nearly all fact files share a common intersection of an item, location, and date as specified above. Such files are expected to come into the platform on a nightly basis and contain that day's transactions or business activity.

Most fact data also supports having currency amounts in their source currency, which is then automatically converted to your primary operating currency during the load process. There are several currency code and exchange rate columns on such interfaces, which should be populated if you need this functionality. The most important ones are shown in the list above, and other optional column for global currencies can be found in the Interfaces Guide. When you provide these fields, they must all be provided on every row of data, you cannot leave out any of the values or it will not load properly.

Here are sample records for commonly used historical load files having a small set of fields populated. These fields are sufficient to see results in RI reporting and move the data to AI Foundation or MFP but may not satisfy all the functional requirements of those applications. Review the Interfaces Guide for complete details on required/optional columns on these interfaces.

SALES.csv:

```
ITEM,ORG_NUM,DAY_DT,MIN_NUM,RTL_TYPE_CODE,SLS_TRX_ID,PROMO_ID,PROMO_COMP_ID,CA
SHIER_ID,REGISTER_ID,SALES_PERSON_ID,CUSTOMER_NUM,SLS_QTY,SLS_AMT_LCL,SLS_PROF
```

```
IT_AMT_LCL,RET_QTY,RET_AMT_LCL,RET_PROFIT_AMT_LCL,TRAN_TYPE,LOC_CURR_CODE,DOC_
CURR_CODE
1235842,1029,20210228,0,R,202102281029,-1,-1,96,19,65,-1,173,1730,605.5,0,0,0,
SALE,USD,USD
1235842,1029,20210307,0,R,202103071029,-1,-1,12,19,55,-1,167,1670,584.5,0,0,0,
SALE,USD,USD
1235842,1029,20210314,0,R,202103141029,-1,-1,30,18,20,-1,181,1810,633.5,0,0,0,
SALE,USD,USD
```

INVENTORY.csv:

```
ITEM,ORG_NUM,DAY_DT,CLEARANCE_FLG,INV_SOH_QTY,INV_SOH_COST_AMT_LCL,INV_SOH_RTL
_AMT_LCL,INV_UNIT_RTL_AMT_LCL,INV_AVG_COST_AMT_LCL,INV_UNIT_COST_AMT_LCL,PURCH
_TYPE_CODE,DOC_CURR_CODE,LOC_CURR_CODE
72939751,1001,20200208,N,0,0,0,104.63,0,48.52,0,USD,USD
73137693,1001,20200208,N,0,0,0,104.63,0,48.52,0,USD,USD
75539075,1001,20200208,N,0,0,0,101.73,0,47.44,0,USD,USD
```

PRICE.csv:

```
ITEM,ORG_NUM,DAY_DT,PRICE_CHANGE_TRAN_TYPE,SELLING_UOM,STANDARD_UNIT_RTL_AMT_L
CL,SELLING_UNIT_RTL_AMT_LCL,BASE_COST_AMT_LCL,LOC_CURR_CODE,DOC_CURR_CODE
89833651,1004,20200208,0,EA,93.11,93.11,53.56,USD,USD
90710567,1004,20200208,0,EA,90.41,90.41,50.74,USD,USD
90846443,1004,20200208,0,EA,79.87,79.87,44.57,USD,USD
```

Fact Data Incremental Logic

Daily or weekly fact data files can be provided incrementally instead of as full snapshots, but the specific handling of incremental changes can be different for the various fact types. The table below summarizes the incremental update logic used on the core fact areas.

Facts	Incremental Logic
Transactional (Sales, Receipts, Markdowns, Adjustments, RTVs, and so on)	Loading transaction data into RAP uses additive merge logic when new data comes into the tables. If the target intersection doesn't exist, it will insert it. If the target intersection DOES exist, then it will merge the records by adding together the source and target fields. For example, an existing sales transaction that is revised will add together the Quantity and Amount fields from the source and target. **Note:** When posting a partial revision, send zeros in fields that should not be adjusted.
Positional (Inventory, Purchase Order, Price, Cost, and so on)	Positional data loaded into RAP must always be for the current date — it cannot be back-posted — and will merge into the target tables with the incoming values (the new day's position is a combination of existing data from yesterday merged with the incoming data). You must send a zero if a given position was moved to zero or dropped from the source system; otherwise it would continue to carry forward the last non-zero position in the database. Refer to the detailed sections later in this chapter for Inventory Position and Pricing examples.

Facts	Incremental Logic
Non-Transactional and Non-Positional Facts (Store Traffic, Flex Facts, History Planning Facts)	Some interfaces that are not related to any transactional or positional data elements, like the Store Traffic or Planning interfaces, use non-additive merge logic. When an existing intersection comes into the staging table, it is merged to the target table but overwrites/replaces the target values with the source values.

Multi-Threading and Parallelism

Due to the high data volumes of most fact data (such as sales and inventory), it is necessary to process the data using multiple CPU threads on the database. In RAP's second-generation architecture, multi-threading is handled automatically. You must not attempt to alter any threading parameters to force a specific thread count greater than 1. If you are providing any data files ending in a `.dat` extension, then it might contain an `ETL_THREAD_VAL` column. This column must be hard-coded to be `1` on all rows without exception; it should not be null and should not be any value greater than 1. Similarly, there are database parameters named `LOC_NUM_OF_THREAD` in the `C_ODI_PARAM_VW` table. These must be set to a value of `1` and should not be altered to any value greater than 1.

Sales Data Requirements

Sales data (`SALES.csv`) operates with the assumption that the source system for the data is an auditing system (like Oracle Sales Audit) or non-Oracle data warehouse system. It applies minimal transformations to the inputs and assumes all the needed cleansing and preparation of transaction data has happened outside of RI. Whether you are sourcing history data from one of those systems or directly from a POS or non-Oracle auditing application, there are some business rules that should be followed.

Requirement	File Type	Explanation
Sales Units	Historical and Ongoing	The values provided for unit quantities should represent the total transaction-line values for an item, split across the gross sales units and return units. In the case of an exchange, you could have both sales and return units on the same line, but most of the time only SLS or RET fields will have a value. These values will be subtracted from each other to form Net Sales Quantity metrics, so they should almost always be positive.
Sales Retail Amounts	Historical and Ongoing	The retail amounts on a sale or return represent the actual selling/return value, after all discounts are subtracted from the base price of the item. In the case of an exchange, you could have both sales and return units on the same line, but most of the time only SLS or RET fields will have a value. These values will be subtracted from each other to form Net Sales Amount metrics, so they should almost always be positive.

Requirement	File Type	Explanation
Sales Profit Amounts	Historical and Ongoing	Profit calculations must take into consideration the cost of the item at the time it was sold, and will vary based on the retailer's costing methods. The standard approach is to use the value for Weighted Average Cost (WAC) multiplied by the units sold/returned, and subtract that total cost value from the retail amount. An item that is sold and later returned may not have the same profit amounts, if the cost has changed between the two transactions or the return was not for the full price of the item. Most POS systems do not track item costs, so providing this data requires an external process to do the calculations for you.
Sales Taxes	Historical and Ongoing	Tax amounts generally represent Value Added Tax (VAT); however this column could be used to capture other tax amounts if loading directly from the POS or external audit system.
Employee Discounts	Historical and Ongoing	These columns are specifically for employee discounts when the employee purchases a product at the POS and gets a special discount (or has the discounted amount returned later on). These values are just the discount amount, meaning the reduction in value from the selling price.
Promotional Discounts	Historical and Ongoing	These values represent the total discount taken off the initial selling price for the line-item in the transaction. These values will almost always be populated for promotional sales. However, a regular or clearance sale could have a further discount applied (like a coupon) and that should also be captured here. These values are used to populate the Sales Promotion fact table for retail type “P” transactions. So make sure that any change in price related to a promotion is included in this discount amount, so that it is copied into other tables for Promotion-level reporting.
Liabilities	Historical and Ongoing	Liabilities are sales that have not yet been charged to the customer, either due to layaway practices or customer orders that are posted as a liability transaction before they are fulfilled. Liabilities are posted with a positive value when incurred, and reversed with a negative value when they are converted into a regular sale or cancelled. Liabilities are a separate set of metrics in RI and do not interact with sales values, as it is expected that the liability will always result in a cancellation or a sale being posted at a later date.
Liability Cancels	Historical and Ongoing	Liabilities that are cancelled should first be reversed and then posted to these fields as a positive amount. A cancelled liability will have no impact on sales and has a separate set of metrics in RI. The retailer can use liability cancellation metrics to track the value of customer orders that were cancelled before being charged.
Retail Type	Historical and Ongoing	The retail type represents the category of sale as one of Regular, Promotion, or Clearance. We use the codes `R/P/C` to denote this value. The priority order when assigning a code is C > P > R, meaning that a clearance sale will always have a clearance type, even if it is also affected by a promotion. This matches the financial practices of RMFCS and Sales Audit, which treat all clearance sales as clearance activity on the stock ledger.

Requirement	File Type	Explanation
Transaction Reversals and Revisions	Historical and Ongoing	When using Sales Audit to audit sales, the export process will automatically handle reversing a transaction and posting revisions to a transaction. Without that, you must manually create a process to send reversals and revisions to the data warehouse matching the same data format. These two records come at the same time. A reversal is an exact opposite of the original transaction line (usually all negative values, unless the original value was negative). This will be added to existing data and zero it out. The revision record should come next and contain the current actual values on the transaction (not the delta or difference in values). Keep in mind that, depending on the method used for calculating sales cost/profit amounts, a reversal and revision may have different values from the original profit amount. This could result in a very small residual profit from the prior revision.
Backposted Sales	Ongoing (Daily)	Sales can be backposted for days prior to the current business date. They will be loaded against their backposted transaction date and aggregated up into the existing sales data for that date. No transformations are done, even if the backposted sale is significantly older (for example,1+ years ago). It will be inserted and aggregated using that past date.
Original Selling Locations	Historical and Ongoing	When including the original selling location of a return transaction, you must also make sure that is a real location included on your Organization input data. Some source systems allow the manual entry of a return's original selling location, so ensure that all such locations are included, or the records will be rejected by the data warehouse.

The columns you provide in the sales file will vary greatly depending on your application needs (for example ,you may not need the sales profit columns if you don't care about Sales Cost or Margin measures). The most commonly used columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction level item or SKU number for the sale and must have a record in the `PRODUCT.csv` file.
ORG_NUM	Must be the store or warehouse number that will get credit for the sale. Note that the location where the sale occurred is not always the location that should be credited for it (for example, a web order placed in-store can still be credited to a web location). Must have a record in `ORGANIZATION.csv` file.
DAY_DT	The date that the transaction occurred in the source system. Must be a valid date within the periods in the `CALENDAR.csv` file.
MIN_NUM	Hour and minute of the transaction in 24-hour, 4-digit format; for example, 11:23 PM would be `2323`. Currently only used in Retail Insights reporting; default to `0` if not needed.

Column Header	Usage
IT_SEQ_NUM	Sequence of a line in the transaction. Every line of a transaction must have its own sequence number. This facilitates uniqueness requirements where the same item could have multiple lines. Without a unique sequence number, the line would not be unique on the input file and the system would see them as duplicate records (duplicate sales lines without a sequence number will be ignored). If you are pre-summing your sales lines such that there will never be duplicate rows, then this column is not needed.
RTL_TYPE_CODE	This is the sales type using one of (`R`, `P`, `C`), where `R` = regular, `P` = promotion, `C` = clearance. This is a critical piece of information for many AIF and Planning purposes. For example, in some AIF modules like SPO, you can choose to include or exclude sales by retail type in the calculations. In forecasting, you can generate different forecasts based on the retail type, which can feed into Planning measures which are split by reg/pro/clr designations.
SLS_TRX_ID	Unique identifier for a sales transaction. By default, it is expected sales data will come at the most granular level (transaction-line). If you are not able to provide true transaction-level data, you can specify some other unique value in this field. This value is part of the business key for the table, so you need to be able to reference the same keys over time (such as when revising or reversing existing transactions).
PROMO_ID PROMO_COMP_ID	This two-part identifier maps to the Promotion and Offer associated with a transaction line. They are required if you wish to load sales by promotion for Retail Insights or certain AI Foundation modules such as Demand Transference. They are also required if you wish to perform promotion lift estimation as part of IPOCS-Demand Forecasting, since the system needs to know the specific promotion linked to a set of sales transactions. When providing these values, also provide the `PROMOTION.csv` file.
CUSTOMER_NUM CUSTOMER_TYPE	Customer identifier on a sales transaction, which is a critical piece of information for many AI Foundation modules, such as Customer Segmentation and Consumer Decisions Trees. `CUSTOMER_TYPE` should not be used at this time, it serves no purpose in downstream modules. If you do include the column, hard-code it to a value of `CUSTID` on all rows, otherwise `CUSTOMER_NUM` values will be ignored.
SLS_QTY SLS_AMT_LCL SLS_PROFIT_AMT_LCL SLS_TAX_AMT_LCL	Represents the gross sales values for the transaction line (meaning before returns). Will almost always be positive, except in cases of negative profit amounts or if you are reversing a prior transaction line to zero out the amounts. Tax amount is for VAT and other special taxes outside the United States (should not be used for US sales tax).
RET_QTY RET_AMT_LCL RET_PROFIT_AMT_LCL RET_TAX_AMT_LCL	Represents customer return transactions. Will almost always be positive, except in cases of negative profit amounts or if you are reversing a prior transaction line to zero out the amounts. Both gross sales and returns are positive because they are subtracted from each other to determine net sales. Tax amount is for VAT and other special taxes outside the United States (should not be used for US sales tax).
REVISION_NUM	If you will be allowing transactions to get revisions from your source system (such as when a sales audit system changes the sales amount after a user audited the transaction) then you should use this field to track the revision number. The initial transaction should come as `1` and later revisions should be posted with a value of `2` or greater. Revision numbers will be stored on the data warehouse table for tracking and auditing purposes.

Column Header	Usage
LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the sales amounts on the transaction. The values in these fields will control how the system converts the amount (such as `SLS_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the point of sale, then `DOC_CURR_CODE` will be the currency of the source system. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single, hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.

Sales Pack Data

If you have pack items (sets of SKUs or multiple units of SKUs sold as one item) then you will need to spread sales of such items from the pack item level to the individual component SKU level. The interface file `SALES_PACK.csv` is provided to load component sales values at a summary level of item/location/date. This data is included in sales calculations for both AI Foundation and Planning applications by adding together the sales of SKUs sold individually on `SALES.csv` and the sales of items inside packs on `SALES_PACK.csv`.

As an example, assume you have a SKU# 1090 which is a white T-shirt. This item is sold individually to customers, but it is also included in a pack of three shirts. The 3-pack is sold using a separate SKU# 3451. You must provide the data for this scenario as follows:

- When SKU 1090 sells to a customer, you will have a transaction for 1 unit on `SALES.csv`
- When SKU 3451 sells to a customer, you will have a transaction for 1 unit on `SALES.csv`, plus a record for SKU 1090 for 3 units on `SALES_PACK.csv` (representing the 3 units inside the pack that sold).

When this data is loaded into other applications like MFP, you will see a total of 4 units of sales for SKU 1090, because we will sum together the sales from both interfaces. The pack-level sale of SKU 3451 is not exported to Planning applications because that would result in double-counting at an aggregate level, but it can be used for other purposes such as Retail Insights reports.

When you are providing `SALES_PACK.csv` you must also provide the pack item/component item relationships using a dimension file `PROD_PACK.csv`. Refer to the *RAP Interfaces Guide* for the full interface specifications of both of these files.

Inventory Data Requirements

Inventory data (`INVENTORY.csv`) has several special requirements that need to be followed when generating historical data and ongoing daily feeds, due to the way the data is stored within Retail Insights as well as the different use-cases in each of the AI Foundation and Planning applications. A retailer may not have the data in the required format in their source system, and adjustments would have to be made in the data extraction process to ensure these rules are followed.

Requirement	File Type	Explanation
Records may be needed before the item/location has any stock on hand	Historical and Ongoing	The inventory position contains fields for inventory movement, such as on-order, in-transit, and reserved quantities. As soon as any of those values may contain data for an item/location (and you intend to use those fields in RAP), a record should be included for inventory, even if the actual stock on hand is still zero that day or week.
Zero balances may be needed for stock on hand	Historical and Ongoing	In general, you are not required to provide zero balance records in history data, just send the non-zero records. The need for zeros in the data falls into one of two use cases: • **Daily Batch**: If no change comes into the system on a given day, we carry forward that balance. This means that you cannot send only non-zero values in the daily data files, as it is assumed the last known value is also the current value for the day or week. You must send a zero balance any time the inventory has moved from non-zero to zero. • **History Load**: In the default load behavior, the history file does not carry balances forward, the sent data is directly inserted. Null values are assumed to be zero in AIF and PDS. This means a zero balance record is unnecessary. However, for integration purposes, a zero balance stock-on-hand may be desired because it is used to determine the Markdown measure classification (clearance or non-clearance promo markdowns). If you switched the load behavior to allow incremental updates, then you must include zero-balance records in the weeks where a position moves from non-zero to zero.
Clearance indicator is used to show the end-of-period status of the item/ location's inventory	Historical and Ongoing	Inventory data has a required column for a Clearance Flag (`Y`/ `N`) to indicate for a given day or week what the status of that item/location's inventory is. The flag is intended to be the end-of-period clearance status of the item/location, so in the history data you should not send multiple records for the same item/location if the flag changed in the middle of the week. Send only one record with the correct flag for the end-of-week value. Default to `N` if you don't use it or don't know it.
Any change to values on the inventory position should send an update of the full record from the source system.	Ongoing (Daily)	If you are using other inventory values besides stock on hand (such as on-order or in-transit), you must ensure the extracts will send a complete record to the inventory interface when any of those values change. For example, a new item/location may carry an on-order balance or in-transit balance for several weeks before it has any stock on hand, so your extracts must trigger changes to those values, not just changes to stock on hand.

For historical loads, when using the default behavior of full weekly snapshots, your load process may work as follows:

1. Generate the first month of week-ending inventory balances in `INVENTORY.csv` for all active item/locations in each week of data. Load using the historical inventory load ad hoc process. Make sure you load Receipts data in parallel with inventory data if you need to capture historical first/last receipt dates against the stock positions (for IPO or LPO usage).
2. Repeat the monthly file generation process, including sets of week-ending balances in chronological order. Remember that you cannot load inventory data out of order, once a given intersection (item/loc/week) is loaded you cannot go back and reload or modify it without deleting it first. Make sure all the requirements listed in the table above are

satisfied for every week of data. Depending on your data volumes you can include more than one month in a single file upload.

3. Load every week of inventory snapshots through to the end of your historical period. If there will be a gap of time before starting nightly batches, plan to load an additional history file at a later date to catch up. Make sure you continue loading Receipts data in parallel with inventory data if first/last receipt date calculations are needed.
4. When you are ready to cutover to batches, you must also re-seed the positions of all item/locations that need to have an inventory record on Day 1 of nightly batch execution (same as for all positional facts in RI). This is needed to fill in any gaps where currently active item/locations are not present in the historical files but need to have an inventory record added on day 1. Use the Seeding Adhoc process for Inventory to do this step, or include a full inventory snapshot file in your first nightly batch run to set all active positions.

When using the incremental load option (`RI_INV_HIST_LOAD_INC = Y` in `C_ODI_PARAM`) the following additional requirements should be considered:

1. Generate data for the first week of history as a full snapshot (the same way you would for the full load method). For every week after that in the same file, only provide the week-ending records where some change occurred. You do not need to provide the same item/locations in the next week if the values are the same as the prior week.
2. If your data volume is sufficiently small, you can generate the entire history data in a single file by having a snapshot in week 1 followed by incremental updates for weeks 2+. If your volumes are still too large to work with, ensure that in each data file, the first week provided is a full snapshot and the following weeks are incremental updates to that snapshot.
3. The rest of the load process is the same as documented for the full snapshot loads, in terms of database configurations and ad hoc POM programs used.

The columns you provide on the inventory file will vary depending on your application needs (for example, you may not need the in-transit or on-order columns if you are only providing data for IPOCS-Demand Forecasting). The most commonly used columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction-level item or SKU number for the inventory position and must have a record in the `PRODUCT.csv` file. Should not be a pack item; all inventory data should be held against the individual components. See Inventory Pack Data if you need to also provide pack item data.
ORG_NUM	Must be the location number that is holding the inventory. Must have a record in the `ORGANIZATION.csv` file. If a location is flagged as non-stockholding in the `ORGANIZATION.csv` file, then it should not have any data in this file.
DAY_DT	The date that the inventory position record is for. In historical files it must be a week-ending date. In nightly batch files it must be the current system business date on all rows; you cannot post inventory for any other date.

Column Header	Usage
CLEARANCE_FLG	Must be a value of `Y` or `N` to indicate the inventory is on clearance or not. Differentiating inventory status is important to RI, AI Foundation, and Planning applications anywhere you would be viewing or planning clearance sales separately from regular sales. Depending on your planning configuration needs, you cannot plan/forecast clearance sales without also having clearance inventory (like when you are specifically forecasting clearance sales and want to compare to clearance inventory). If you are not implementing a solution that separates regular and clearance activities, then you would set this value to `N` on all rows.
INV_SOH_QTY INV_SOH_COST_AMT_LCL INV_SOH_RTL_AMT_LCL	Stock on hand and available to sell. The cost and retail amounts are the total cost/retail value of all units of stock. All the values on this interface are close-of-business values (for example, stock on hand at end of day or end of week). Pack item inventory should be broken down into their component quantities and amounts and summed together with the stock for the same SKUs held individually. Only one row should be provided per item/location/date.
INV_IN_TRAN_QTY INV_IN_TRAN_COST_AMT_LCL INV_IN_TRAN_RTL_AMT_LCL	Stock in-transit between two owned locations (such as from warehouse-to-store or store-to-store). This stock is shipped but not yet received. It will be summed together with SOH values for MFP to show total owned stock for a location, inclusive of units that will arrive at that location soon. All the same criteria listed for SOH values apply to these fields as well.
INV_UNIT_RTL_AMT_LCL	The base unit retail value of an item at the location for this stock position. Used by multiple applications to display the retail value for a specific item/location in the context of owned inventory. May or may not be the same value as the Selling Price provided on the `PRICE.csv` file depending on how the business calculates the retail value of owned inventory. Other columns like `INV_SOH_RTL_AMT_LCL` should be a multiplication of this value times the unit quantity.
INV_AVG_COST_AMT_LCL INV_UNIT_COST_AMT_LCL	The average cost and unit cost of owned inventory for this item/location. The default cost used by retailers is usually the average cost (also known as Weighted Average Cost or WAC). Other columns like `INV_SOH_COST_AMT_LCL` should be a multiplication of one of these values times the unit quantity.
LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the inventory data. The values in these fields will control how the system converts the amount (such as `INV_SOH_COST_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.

Inventory Pack Data

If you have pack items (sets of SKUs or multiple units of SKUs sold as one item) then you will need to separately provide the inventory for those items. The interface file `INVENTORY_PACK.csv` is provided to load pack item inventory at a level of pack item, location, and date. The main inventory file must not include any pack items: it is only for regular items and component items (which are broken down from pack item data). Pack item inventory is generally tracked only at warehouses when the item is first received from a vendor. Once the pack is allocated or shipped to stores, it will be tracked by its individual component items on `INVENTORY.csv`, not by the original pack item number. Pack inventory may be used within some Inventory Planning Optimization modules, if you need those applications to be aware of warehouse inventory of pack items for the purposes of purchase order or allocation creation. It may also be used in Retail Insights reporting if you need to report on the warehouse inventory at a pack item level. No other solution currently uses this data because all other operations are performed on the pack component items only.

The requirements for providing pack item inventory are similar to the main inventory file. It is a positional fact with quantity, cost, and retail columns for the major inventory areas of on-hand, in-transit, and on-order totals. Pack items that are not sellable may not have any values in the retail (`RTL`) columns, which is normal and expected. The history data requirements are also the same, such as requiring that it be loaded as full weekly snapshots in chronological order. The history load programs are included in the same processes as the main inventory fact load. Only RI requires history data for reporting; it would be sufficient to load just the current daily data for use in IPO.

Price Data Requirements

Pricing data (`PRICE.csv`) has several special requirements that need to be followed when generating historical data and ongoing daily feeds, due to the way the data is stored within Retail Insights as well as the different use-cases in the RI and AI Foundation applications. A retailer may not have the data in the required format in their source system, and adjustments would have to be made in the data extraction process to ensure these rules are followed.

Requirement	File Type	Explanation
The first price file must include records for all active item/ locations	Historical	The pricing fact is stored positionally, meaning that it first needs a starting position for an entity (for example, an initial price for an item/location) and then it may not be sent again unless there is a change to the value. The very first price file loaded into the data warehouse must contain a starting position for all active item/locations. How you determine which item/locations were active on that day in history will depend on your source system (for example, you can base it on items having stock on hand, historical ranging status, or a price history table if you have one).
The first record sent for an item/ location must come as a new price transaction type	Historical and Ongoing	Price records have a column for the price type (`PRICE_CHANGE_TRAN_TYPE`) with a fixed list of possible values. The value `0` represents a new price, which means it is the first time our system is getting a price for this item/ location. All item/locations must be given a type `0` record as their first entry in the historical or ongoing data files. All the initial position records in the first file will have `type=0`. Also, all new item/locations coming in later files must first come with `type=0`.

Requirement	File Type	Explanation
Price records should follow a specific lifecycle using the type codes	Historical and Ongoing	The typical flow of price changes that will occur for an item/ location should be as follows: • New price/cost (PRICE_CHANGE_TRAN_TYPE=0) • Regular cost changes (PRICE_CHANGE_TRAN_TYPE=2) • Regular price changes (PRICE_CHANGE_TRAN_TYPE=4) • Promotional/temporary markdowns (PRICE_CHANGE_TRAN_TYPE=9) • Clearance markdowns (PRICE_CHANGE_TRAN_TYPE=8) • Price reset due to new selling cycle or season change (PRICE_CHANGE_TRAN_TYPE=0) An item/location may have many changes with types 2/4/8/9 before eventually staying at 8 (for a final markdown) or resetting to 0 (if the item lifecycle should restart).
Price changes are for the end-of-day value only	Historical and Ongoing	An item price may change many times a day, but you must only send the end-of-day final position for the item/location. The file interface assumes only one record will be sent per item/location/effective date, representing the final price on that date.

For historical loads, this results in the following flow of data across all your files:

1. Generate an initial position `PRICE.csv` that has all `type=0` records for the item/locations you want to specify a starting price for. Load this as the very first file using the historical load ad hoc process.
2. Generate your first month of price change records. This will have a mixture of all the price change types. New item/location records may come in with `type=0` and records already established can get updates using any of the other type codes. Only send records when a price or cost value changes; do not send every item/location on every date. You also must not send more than one change per item/location/date.
3. Repeat the monthly file generation (or more than one month if your data volume for price changes is low) and load process until all price history has been loaded for the historical timeframe.
4. When you are ready for the cutover to batches, you must also re-seed the positions of all item/locations that need a price record on Day 1 of nightly batch execution (same as for all positional facts in RI). This is needed to fill in any gaps where currently active item/ locations are not present in the historical files, but need a price record added on day 1. Use the Seeding Ad Hoc process for Pricing to do this step, not the historical load.

In most cases, you will be providing the same set of price columns for any application. These columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction-level item or SKU number and must have a record in the `PRODUCT.csv` file. You should provide a price record for all sellable or inventoried items.
ORG_NUM	Must be the location number where the item on the record is ranged to. Must have a record in `ORGANIZATION.csv` file. A price should be provided for every location where a transaction could occur for the item on the record, such as a sale or return.

Column Header	Usage
DAY_DT	The date that the price record is for. In historical files it will be the effective date of the price change. In nightly batch files it must be the current system business date on all rows, you cannot post prices for any other date. The data sent nightly is for price changes effective for that one date.
PRICE_CHANGE_TRAN_TYPE	The type of price change event, represented by a numerical code as defined in the business rules earlier in this section. This is NOT part of the primary key, meaning that you can only provide a single price per item/location/date.
SELLING_UNIT_RTL_AMT_LCL	The current selling retail price of the item, at the specified location, on the specified business date. The selling price will generally reflect the current "ticket price" of the item that a customer would pay before transaction-level adjustments like coupons or loyalty awards. The price is also a key input to certain forecast functions (LLC, causal, promo lift).
BASE_COST_AMT_LCL	The unit cost of the item at this location. Only used by AI Foundation and Retail Insights reporting. In LPO, the price and cost are both used to determine the Gross Margin amount for the item/location in pricing objectives.
LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the price data. The values in these fields will control how the system converts the amount (such as `SELLING_UNIT_RTL_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.
ORIG_SELLING_RTL_AMT_LCL	This is a nullable column provided for specifying the original full price of an item/location in cases where you need to update the original price as part of a different price change posting. For example, an item may begin selling at a store only after it has been marked down. The first price change event is a markdown (type 8) and you also populate this field on the same row to indicate what the original price was prior to the markdown.
LST_REG_RTL_AMT_LCL	This is a nullable column provided for specifying the most recent regular price of an item/location in cases where you need to update the regular price as part of a different price change posting. For example, an item may begin selling at a store only after it has been marked down. The first price change event is a markdown (type 8) and you also populate this field on the same row to indicate what the last regular price was prior to the markdown.

Receipts Data Requirements

Receipts data (`RECEIPT.csv`) is used specifically for receipt transactions where inventory units are received into an owned location (like a store or warehouse), and that receipt impacts the stock on hand for the location. The file is used for several purposes throughout RAP: it is needed by MFP for inventory plans, by IPO and LPO for determining first and last receiving dates by item/location, and by RI for reporting on receipt activity. The receipts data must be loaded in parallel with inventory position if AIF modules are being implemented, because the calculations for IPO/LPO are done up front during each load of inventory position and receipt files.

Rule	Explanation
Receipt Types	The receipts are provided using a type code, with 3 specific codes supported: • **20** – This code is for purchase order receipts, which are usually shipments from a supplier into a warehouse (but can be into stores). • **44~A** – These are allocation transfer receipts resulting from allocations issued to move warehouse inventory down to stores. The receipt occurs for the store location on the day it receives the shipment. • **44~T** – These are generic non-allocation transfer receipts between any two locations. MFP GA solution only uses type 20 transactions but the rest of the RAP solutions use all types.
Receipts vs. Transfers	Transfer receipts are not the same thing as transfers (`TRANSFER.csv`) and both datasets provide useful information. Transfer receipts are specific to the receiving location only and occur at the time the units arrive. Transfers are linked to both the shipping and receiving locations, and they should be sent at the time the transfer is initiated. The MFP GA solution receives transfers from the `TRANSFER.csv` file only, but the other solutions will want both `RECEIPT.csv` and `TRANSFER.csv` files to have the transfer-related data.
Unplanned Receipts	It is possible for a location to receive inventory it did not ask for (for example, there is no associated PO or allocation linked to those units). Such receipts should still appear as a type 44~T receipt transaction, so long as those units of inventory do get pulled into the location's stock on hand.

In most cases, you will be providing the same set of receipt columns for any application. These columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction level item or SKU number and must have a record in the `PRODUCT.csv` file.
ORG_NUM	Must be the location number where the item is received. Must have a record in `ORGANIZATION.csv` file.
DAY_DT	The date that the receipt occurred on. Receipts can occur on any date both in history and nightly batch files.
INVRC_TYPE_CODE	Indicates the type of receipt into a location using merchandising transaction codes `20` (purchase order receipt), `44~A` (allocation transfer receipt), or `44~T` (non-allocation transfer receipt). Only PO receipts are used in MFP, as they represent actual inventory entering the company, and not just movement between two owned locations.
INVRC_QTY INVRC_COST_AMT_LCL INVRC_RTL_AMT_LCL	The units being received and the total cost/retail value of those units relative to the receiving location.

Column Header	Usage
LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the receipt data. The values in these fields will control how the system converts the amount (such as `INVRC_COST_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.

Transfer Data Requirements

Transfer data (`TRANSFER.csv`) is used to capture movement of inventory between two locations (warehouses or stores). Transfer transactions are sent at the time the transfer is initiated at the shipping location. Transfer transactions are used primarily in Planning applications and RI reporting.

Rule	Explanation
Transfer Types	Transfers are provided using a type code, with 3 specific codes supported: • **N** – Normal transfers are physical movement of inventory between two locations that impacts the stock on hand • **B** – Book transfers are financial movement of inventory in the system of record that doesn't result in any physical movement, but still impacts the stock on hand • **I** – Intercompany transfers involve inventory moved into or out of another location that is part of a different legal entity, and therefore the transfer is treated like a purchase transaction in the source system Most transfers are categorized as Normal (`N`) by default. All transfer types are sent to Planning but would be loaded into separate measures as needed based on the type. Because transfers and receipts are separate measures used for different purposes, there is no overlap despite having similar information in both files.
Transfer In vs. Transfer Out	The transfers file has two sets of measures for the unit/cost/retail value into the location and out of the location. Typically these values contain the same data, but since they are aggregated and displayed separately in the target systems, they are also separate on the input so you have full control over what goes into each measure. For example, a transfer in of 5 units to location 102 would also have a transfer out of 5 units leaving location 56 (on the same record).

In most cases, you will be providing the same set of transfer columns for any application. These columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction level item or SKU number and must have a record in the `PRODUCT.csv` file.
ORG_NUM	Must be the location number where the item inventory is being shipped to (i.e. the receiving location). Must have a record in `ORGANIZATION.csv` file.

Column Header	Usage
DAY_DT	The date that the transfer was initiated on (NOT the date when it completes or is received). Transfers can occur on any date both in history and nightly batch files.
FROM_ORG_NUM	Must be the location number where the item inventory is being moved out of (that is, the shipping location). Must have a record in `ORGANIZATION.csv` file.
TSF_TYPE_CODE	The numerical code representing the transfer type, as described in the business rules for transfers earlier in this section. This is a part of the primary key, meaning you may have multiple records for the same item/location/from-location/date with different transfer type codes.
TSF_TO_LOC_QTY TSF_TO_LOC_COST_AMT_LCL TSF_TO_LOC_RTL_AMT_LCL	The units and total cost/retail values for the transfer relative to the “to” location (`ORG_NUM`). Will be separately aggregated and displayed in MFP.
TSF_FROM_LOC_QTY TSF_FROM_LOC_COST_AMT_LCL TSF_FROM_LOC_RTL_AMT_LCL	The units and total cost/retail values for the transfer relative to the “from” location (`FROM_ORG_NUM`). Will be separately aggregated and displayed in MFP.
LOC_CURR_CODE FROM_DOC_CURR_CODE FROM_LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the transfer data. The values in these fields will control how the system converts the amount (such as `TSF_TO_LOC_COST_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. The `FROM` columns will be the same thing applied for the `FROM_ORG_NUM` location. Review the section on Exchange Rate dimension data for additional scenarios.

Adjustment Data Requirements

Adjustments data (`ADJUSTMENT.csv`) is used to capture manual changes to stock on hand made for any reason that does not fall into one of the other categories like sales or receipts. Adjustments are used only in Planning and RI applications.

Rule	Explanation
Adjustment Types	The adjustments are provided using a type code, with 3 specific codes supported: • **22** – These adjustments are your standard changes to inventory for wastage, spoilage, losses, and so on. In Planning they are categorized as Shrink adjustments. • **23** – These adjustments are for specific changes that impact the Cost of Goods Sold but are not an unplanned shrink event, such as charitable donations. In Planning they are categorized as Non-Shrink adjustments. • **41** – These adjustments are targeted to reporting needs specifically and are the result of a stock count activity where the inventory levels were already adjusted in the store's inventory as part of the count, but you want the adjustment captured anyway to report against it Only types `22` and `23` go to Planning applications. Type `41` is used within RI for reporting.
Reason Codes	Reason codes are used to identify the specific type of adjustment that occurred for that item, location, and date. If you are loading data for Planning apps, then they are not required because Planning apps do not look at reason codes. They are only used for RI reporting. There are no required codes; it will depend on the data in your source system. The codes should be numerical, and there is a Description field that must also be provided for the display name.
Positive and Negative Values	Adjustments should be positive by default. A positive adjustment on the input file means a decrease in the stock on hand at the location. A negative adjustment means an increase to stock on hand (basically you have adjusted the units back into the location's inventory, which is less common). When the data is sent to MFP, the default planning import will invert the signs for the positive adjustments to become subtractions to inventory.

In most cases, you will be providing the same set of adjustment columns for any application. These columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction level item or SKU number and must have a record in the `PRODUCT.csv` file.
ORG_NUM	Must be the location number where the item inventory was adjusted. Must have a record in `ORGANIZATION.csv` file.
DAY_DT	The date that the adjustment occurred on. Adjustments can occur on any date both in history and nightly batch files.
INVADJ_TYPE_CODE	Indicates the type of adjustment to inventory using transaction codes `22` (shrink adjustment), `23` (non-shrink adjustment), or `41` (stock count adjustment). The codes determine the associated Planning measures the data is placed into. Code `41` is only used in Retail Insights reporting.
INVADJ_QTY INVADJ_COST_AMT_LCL INVADJ_RTL_AMT_LCL	The units and total cost/retail values for the adjustment. The cost and retail amounts may have varying calculations in your source system depending on how shrink and non-shrink inventory adjustments are determined. Adjustments can be negative or positive depending on whether inventory is being removed or added to the stock on hand. The sign is reversed by MFP, meaning negative amounts on the input will display as positive values on the MFP measures.

Column Header	Usage
LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the adjustment data. The values in these fields will control how the system converts the amount (such as `INVADJ_COST_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.

RTV Data Requirements

Return to vendor data (`RTV.csv`) is used to capture returns of inventory from your stores or warehouses back to the original vendor. An RTV is a transaction that decreases your owned inventory at a location because you are shipping units to a non-owned location. RTVs are used only in Planning and RI applications.

Rule	Explanation
Supplier IDs, Reason Codes, Status Codes	All of the reason code, supplier number, and status code fields in an RTV record are optional and used only for RI reporting purposes, because planning applications do not report at those levels. If you are not specifying these values, leave the columns out of the file entirely, and a default value of -1 will be assigned to the record in those columns.
Positive and Negative Values	RTV transactions should always be positive values. Only send negative values to reverse a previously-sent transaction in order to zero it out from the database.

In most cases, you will be providing the same set of RTV columns for any application. These columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction level item or SKU number and must have a record in the `PRODUCT.csv` file.
ORG_NUM	Must be the location number where the RTV is being shipped out from. Must have a record in `ORGANIZATION.csv` file.
DAY_DT	The date that the RTV occurred on. RTVs can occur on any date both in history and nightly batch files.
RTV_QTY RTV_COST_AMT_LCL RTV_RTL_AMT_LCL	The units and total cost/retail values for the returns to vendor.

Column Header	Usage
LOC_CURR_CODE DOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the RTV data. The values in these fields will control how the system converts the amount (such as `RTV_COST_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.

Markdown Data Requirements

Markdown data (`MARKDOWN.csv`) is used to capture changes in retail value of owned inventory due to a permanent or temporary price change. Markdowns are used only in Planning and RI applications. There are separate measures on the input file for markdown and markup effects depending on the kind of price change and direction of the change. For regular and clearance price changes, the file captures the total change in value of owned inventory units on the day the price change goes into effect. For promotional or temporary price changes, the file should have only the marginal effects of the price change when any of that inventory is sold to a customer (since the overall value of your inventory is not changed by a temporary promotion).

Rule	Explanation
Markdown Amounts	Markdown amounts are only the change in total value of inventory, not the total value itself. Permanent and clearance price changes result in markdown amounts derived like this: Markdown Retail = (SOH*Old Retail) – (SOH*New Retail) Markdown Retail = (150*15) – (150*12) = $450 Promotional price changes do not need the total markdown amount calculation, and instead send a promotion markdown amount at the time of any sale: Promotional Markdown Retail = (Units Sold*Old Retail) – (Units Sold*New Retail) Promotional Markdown Retail = (5*17) – (5*15) = $10 Markdown amounts will generally be positive values when the price was decreased, and the target systems will know when to add or subtract the markdown amounts where needed.
Markdown Types	The markdowns are provided using a type code, with 3 specific codes supported: • **R** – Regular permanent price changes that are not considered a clearance price • **C** – Clearance markdowns which are permanent and intended to be used at end-of-life for the item • **P** – Promotional markdowns which are temporary price changes or discounts that are limited to a period of time
Markup Handling	When a regular price is increased or a clearance price is set back to regular price, you can send a separate transaction with positive Markup values populated in the record. You do not need to send negative values to reverse a markdown; the target systems can use the markup measures to do that. A similar rule applies to the markdown/markup cancel measures.

Rule	Explanation
Inventory Usage for PDS Measures	Markdown data is joined with inventory data when you are exporting it to Planning applications, specifically to calculate two markdown measures (reg-promo and clearance-promo markdown amounts). The markdown export uses the clearance flag from the inventory history to determine the measure rollups. If there is no inventory record for a given item/loc/week intersection, the markdown data will default into the reg-promo markdown measure.

In most cases, you will be providing the same set of markdown columns for any application. These columns are listed below with additional usage notes.

Column Header	Usage
ITEM	Must be the transaction level item or SKU number and must have a record in the `PRODUCT.csv` file.
ORG_NUM	Must be the location number where the markdown transaction occurred. Must have a record in `ORGANIZATION.csv` file.
DAY_DT	The date that the markdown occurred on. Markdowns can occur on any date both in history and nightly batch files.
RTL_TYPE_CODE	The type of markdown using one of `R`/`P`/`C` characters to identify it as described in the business rules above. The type code determines which measures in Planning will get the data. Regular and clearance markdowns are considered as "inventory devaluation" while promo markdowns are shown as "markdowns". Promo markdowns are further split into clearance-promo and reg-promo using the clearance flag from Inventory Position data.
MKDN_QTY MKDN_AMT_LCL	The units affected by a markdown and the total change in retail value as a result of a markdown. Both values will be positive numbers when representing a decrease in retail value as the result of a markdown.
MKUP_QTY MKUP_AMT_LCL	The units affected by a markup and the total change in retail value as a result of a markup. Both values will be positive numbers when representing an increase in retail value as the result of a markup.
MKDN_CAN_QTY MKDN_CAN_AMT_LCL	The units affected by a cancelled markdown and the total change in retail value as a result of the cancellation. Both values will be positive numbers when representing an increase in retail value as the result of the cancellation.
MKUP_CAN_QTY MKUP_CAN_AMT_LCL	The units affected by a cancelled markup and the total change in retail value as a result of the cancellation. Both values will be positive numbers when representing a decrease in retail value as the result of the cancellation.
DOC_CURR_CODE LOC_CURR_CODE	The currency codes linked to the retail amounts on the markdown data. The values in these fields will control how the system converts the amount (such as `MKDN_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.

Purchase Order Data Requirements

Purchase order data (ORDER_HEAD.csv and ORDER_DETAIL.csv) is used to capture the raw PO details at the lowest level. Purchase orders are used in Planning, IPO, and RI applications. For Planning, the PO data is aggregated from the order level to a forward-looking summary based on the open-to-buy (OTB) week date. The raw details are used as-is in RI and IPO.

Rule	Explanation
Daily Data Requirements	It is expected that the PO header and detail files start as full daily snapshots of all active or recently closed orders. The detail data is maintained positionally so that, if no update is received, we will continue to carry forward the last known value. Once daily batches have started, you can transition the PO details file only to an incremental update file (header file must always be a complete snapshot). When sending data incrementally, you must include all order updates for a given date, both for open and closed orders. If an order changed at the header level (such as closing or cancelling the order), you should send all the detail lines in that order even if some didn't change. This includes when order lines are fully received and move to 0 units remaining, these changes must be sent to RAP. If you are unable to satisfy these incremental data requirements, you may change the parameter PO_FULL_LOAD_IND to Y to instead provide full snapshots of only non-zero order lines and the system will zero out the rest of the orders automatically.
Historical Data and Past Dates	The Order interfaces do not support loading historical data or data with past dates on the detailed order-line records. Every time you load orders, it is for the current set of data for a single business date. The DAY_DT value on the detail file should be the same on all rows and be set to the business date the data is for. You also cannot reload the same date multiple times; the detail table follows the rules for positional facts as described in Positional Data Handling section below.
Order Status	The header file for POs has a variety of attributes but one of the most important is the status, which should be either A (active) or C (closed). Active orders are used in the PO calculations. When sending daily PO files, you must include both active and closed order updates, because we need to know an order has been completed so it can stop being included in calculations.
OTB EOW Date	The OTB end-of-week date is used for the Planning aggregations to create a forward-looking view of expected receipts from POs. Open order quantities are aggregated to the OTB week before being exported to Planning. If the OTB week has elapsed, the order quantities are included in next week's OTB roll-up regardless of how far in the past the date is, because the earliest that PO can come as receipts is in the next business week.
Include On Order Indicator	There is a required flag on the order header file to tell the system that an order should be included in calculations for Planning or not. When the flag is set to Y, that order's details will be used for the aggregated on-order values. If set to N, the order details will not be used (but will still be present in the database for other purposes like RI reporting and inventory optimization).
Orders of Pack Items	Purchase orders placed for pack items need to be spread down to their component item level. This is necessary to keep the PO data consistent with the inventory and transactional data, which is also only held at the component item level. If you want to maintain the original pack item information for a given PO detail line, separate fields are provided at the end of the ORDER_DETAIL.csv file for this purpose. The pack item fields on the input file will be converted to pack level order data for use in the IPO Lifecycle Allocation & Replenishment application.

The ORDER_HEAD.csv and ORDER_DETAIL.csv files both have a minimum set of required fields to make the integrations within RAP function, so those will be listed out below with additional usage notes. The two files are tightly coupled and it's expected that you send both at the same time; you will never send only one of them.

Table 8-10 ORDER_HEAD.csv

Column Header	Usage
ORDER_NO	Unique identifier of a purchase order in the source system. This must be the same number used across the entire life of the order, so that you can post updates and revisions to it over time. If your source system changes the order number for any reason, be aware that you may need to keep track of old order numbers to post the updates to RAP for the right orders.
STATUS	Tells the system if the order is active (A) or closed/cancelled (C). Only active orders will be used for Planning applications, even if a closed order still has on-order quantities on the records. It is important that you post closed orders to RAP when they close, so that we have an accurate status for all orders.
OTB_EOW_DATE	The week-ending date where the open order quantities should be considered for Open To Buy (OTB) planning. This will drive the aggregation of the purchase order data into future weeks before it is loaded into Planning applications. When the OTB_EOW_DATE is in the past, any remaining open order quantities will be pushed into the next possible week-ending date that is in the future, because those units cannot be received before that time.
INCLUDE_ON_ORDER_IND	A flag to instruct the system on which orders should be considered for Open to Buy and any other open on-order calculations. If the flag is N then the order will not be sent to Planning, but it can still be used in Retail Insights reporting or custom extensions.

Table 8-11 ORDER_DETAIL.csv

Column Header	Usage
ITEM	Must be the transaction level item or SKU number and must have a record in the PRODUCT.csv file. Should not be a pack item, even if the supplier would be delivering in packs. It is expected that inventory and purchase order data are always at the component item level. Pack item data should be spread to their components before sending this file. To preserve the original pack information, separate columns in the file can be used.
ORG_NUM	Must be the location number where the order was placed from and will be received at. Must have a record in ORGANIZATION.csv file. It is usually a warehouse but can be a store if you allow direct-to-store deliveries.
DAY_DT	The current business date in the system for this data load. You cannot send order data for any other date except the current business date, as this is a positional table like Inventory. Past/future dates will not be accepted. The purpose of this column is mainly for reference and archival purposes (for example, when looking at old data files you will know which business date this set of records was for).
ORDER_NO	A cross-reference to the order number on ORDER_HEAD.csv.

Table 8-11 (Cont.) ORDER_DETAIL.csv

Column Header	Usage
PO_ONORD_QTY PO_ONORD_COST_AMT_LCL PO_ONORD_RTL_AMT_LCL	The current outstanding order quantities and total cost/retail value of the units on order. These values represent the expected units that have not been received yet. Because this interface is positional (like inventory) we will carry forward the last known quantities every day unless you update the system with new records. This means that, even when the order is received or cancelled, we must eventually get a record that moves all of these columns to 0, or we will carry forward the non-zero values forever.
DOC_CURR_CODE LOC_CURR_CODE	The currency codes linked to the cost and retail amounts on the order data. The values in these fields will control how the system converts the amount (such as `PO_ONORD_COST_AMT_LCL`) and how the `EXCH_RATE.csv` file data will be used. If you are providing the data in the source currency from the store or warehouse location, then `DOC_CURR_CODE` will be the currency of the source location. If the data is already all on one currency, then `DOC_CURR_CODE` will be that currency code. `LOC_CURR_CODE` is the primary reporting currency you wish RAP applications to operate in, so it will generally be a single hard-coded value on all your data files. Review the section on Exchange Rate dimension data for additional scenarios.
PO_PACK_NUM PO_PACK_ONORD_QTY PO_PACK_ORDERED_QTY PO_PACK_RECEIVED_QTY PO_PACK_ITEM_UNIT_COST PO_PACK_ITEM_UNIT_RETAIL	Use these columns to provide the original values for a purchase order of a pack item. The order lines should be for the component items within the pack, but these fields may reference the original pack number, quantities, and amounts from the order. If the order is for a complex pack with multiple components, then the same values might be repeated on these columns for all items in a specific pack. This will be accounted for when exporting the data to IPO.

It's also necessary to understand the lifecycle of a purchase order and how that should be reflected in the data files over time. RAP will require data to be sent for each step in the order process as outlined below.

1. When the order is approved, the `ORDER_HEAD` file should contain a row for the order with `status=A` and the `ORER_DETAIL` should contain the items on the order with non-zero quantities for the on-order amounts.
2. As the lines of the order are received, `ORDER_HEAD` should continue to have the row for the order with every update, and `ORDER_DETAIL` should be sent with the order lines that require changes from the last known value. If you have the ability to detect which order lines changed, you only need to send those. RAP will remember and carry forward any order lines that were not updated. If you can't detect the changes to order lines, just send all lines in the order every time.
3. If any lines are cancelled from the order, you must send that update as a set of zero values on the `PO_ONORD_*` columns in `ORDER_DETAIL` to zero out the cancelled lines in RAP. Similarly, if the entire order is canceled or closed before being fully received, you must send all lines of the order with zero values on the `PO_ONORD_*` columns in `ORDER_DETAIL` and also update `ORDER_HEAD` to have a status of `C`. If this is not possible in your source system, you must configure the parameter `PO_FULL_LOAD_IND` to a value of `Y` in **Manage System Configurations**, then you will be allowed to send full loads of only non-zero order lines and the system will zero out the rest.

4. As order lines start to be received normally, send the new order quantities for each change, including when a line is fully received and moves to 0 units on order. When an order becomes fully received we need all rows of data in RAP to move to 0 for that order's values, so that we stop including it in future on-order rollups. If `PO_FULL_LOAD_IND=Y` then we don't need zero balance updates from you, just stop sending the order details when it reaches zero and we will zero it out automatically.
5. When an order is finally fully-received and closed, send one final update where `ORDER_HEAD` shows the status as `C` and the `ORDER_DETAIL` data is moved to 0 units on order for any lines not updated yet.

Depending on your source system, it can be difficult to detect all of these changes to the purchase orders over time and send only incremental updates. In such cases, you may always post all orders to RAP which are active or have been closed within recent history and we will merge the data into the system on top of the existing order records. Then the main requirement that must be accounted for is the cancelling or removal of order lines from an order, which must still be tracked and sent to RAP even if your source system deletes the data (unless `PO_FULL_LOAD_IND=Y`).

Other Fact File Considerations

The following section describes additional data considerations that may apply to your implementation depending on the types and volumes of data being provided to the platform. Review each topic closely, as it affects the data provided in the foundation files.

Positional Data Handling

The largest sets of fact data in the platform tend to be those that represent every possible item/location combination (such as prices or costs). To efficiently store and process these data volumes, a data warehouse technique known as compression is used to capture only the changed records on a day-to-day basis, effectively maintaining a "current position" for every set of identifiers, which is updated during each batch execution. The output of this compression process is called positional data, and the following functional areas use this method of data load and storage:

- Inventory (INV, INVPK, and INVU)
- Prices (PRICE)
- Costs (BCOST and NCOST)
- Purchase Orders (PO_ONORD) and Allocations On Order (PO_ONALC)

Positional data loads follow very specific rules and cannot be processed in the same manner as non-positional data such as sales transactions.

Table 8-12 Positional Data Rules

Rule	Explanation
Data Must be Sequential	Positional data must be loaded in the order of the calendar date on which it occurs and cannot be loaded out-of-order. For example, when loading history data for inventory, you must provide each week of inventory one after the other, starting from Week 1, 2, 3, and so on.
Data Cannot be Back Posted	Positional data cannot be posted to any date prior to the current load date or business date of the system. If your current load date is Week 52 2021, you cannot post records back to Week 50: those past positions are unable to be changed. Any corrections that need to be loaded must be effective from the current date forward.

Table 8-12 (Cont.) Positional Data Rules

Rule	Explanation
Data Must be Seeded	Because positional data must maintain the current position of all data elements in the fact (even those that are inactive or not changing) it is required to initialize or “seed” positional facts with a starting value for every possible combination of identifiers. This happens at two times: 1. The first date in your history files must be full snapshots of all item/locations that need a value, including zero balances for things like inventory. 2. Special seed programs are provided to load initial full snapshots of data after history is finished, to prepare you for nightly batch runs. After seeding, you are allowed to provide incremental datasets (posting only the positions that change, not the full daily or weekly snapshot). Incremental loads are one of the main benefits of using positional data, as they greatly reduce your nightly batch runtime.

Throughout the initial data load process, there will be additional steps called out any time a positional load must be performed, to ensure you accurately capture both historical and initial seed data before starting nightly batch runs.

System Parameters File

The dimension file for `RA_SRC_CURR_PARAM_G.dat` is not used as part of your history load process directly, but instead provides an important piece of information to the platform for operational activities. This file must contain the current business date associated with the files in the ZIP package. The file should be included with the nightly ZIP upload that contains your foundation data, such as `RI_RMS_DATA.zip` or `RAP_DATA.zip`. You may use a CTX file to change the column delimiter used in the file, but you should retain the other format characteristics exactly as shown below (for example, the date format must be `YYYYMMDD` and it does not need a header row).

The file has only two generic columns, `PARAM_NAME` and `PARAM_VALUE`. When data is sourced from RMFCS, it will be automatically generated and sent to RAP in the nightly batch. If your data does not come from RMFCS, then you need to include the file manually. Currently, only two rows will be used, but future releases may look for additional parameters in the file.

For non-MFCS customers, the file should contain at least the following rows:

```
VDATE|20220101
PRIME_CURRENCY_CODE|USD
```

For anyone that will be using MFCS now or at any time in the future, you instead should provide the full set of parameters that MFCS would eventually be generating for you, like so:

```
PRIME_CURRENCY_CODE|USD
CONSOLIDATION_CODE|C
VAT_IND|Y
STKLDGR_VAT_INCL_RETL_IND|Y
MULTI_CURRENCY_IND|Y
CLASS_LEVEL_VAT_IND|Y
DOMAIN_LEVEL|D
CALENDAR_454_IND|4
```

```
VDATE|20230506
NEXT_VDATE|20230507
LAST_EOM_DATE|20240131
CURR_BOM_DATE|20240201
MAX_BACKPOST_DAYS|10
PRIME_EXCHNG_RATE|1
PRIMARY_LANG|EN
DEFAULT_TAX_TYPE|GTAX
INVOICE_LAST_POST_DATE|20170101
MERCH_COST_MAP_CODE|9999999
```

The parameter value with `VDATE` is the current business date that all your other files were generated for in `YYYYMMDD` format. The date should match the values on your fact data, such as the `DAY_DT` columns in sales, inventory, and so on. This format is not configurable and should be provided as shown. The parameter value with `PRIME_CURRENCY_CODE` is used by the system to set default currencies on fact files when you do not provide them yourself or if there are null currency codes on a row.

For the data sample above, here are the CTX file parameters you must provide in the `RA_SRC_CURR_PARAM_G.dat.ctx` file:

```
#TABLE#RA_SRC_CURR_PARAM_G#
#DELIMITER#|#
#DATEFORMAT#YYYYMMDD#
#REJECTLIMIT#1#
#RECORDDELIMITER#\n#
#IGNOREBLANKLINES#false#
#SKIPHEADERS#0#
#TRIMSPACES#rtrim#
#TRUNCATECOL#false#
#COLUMNLIST#
PARAM_NAME
PARAM_VALUE
```

Assuming you will be using RDE jobs to extract data from MFCS later on, the other parameters can be provided as shown above or with any other values. The first time you run the RDE job `ETLREFRESHGENSDE_JOB`, it will extract all the parameter values from MFCS and directly update the `RA_SRC_CURR_PARAM_G` table records. The update from MFCS assumes that `RA_SRC_CURR_PARAM_G` already has rows for all of the above parameters, which is why it is important to initialize the data as shown if you are loading data from flat files.

9 Extensibility

The Retail Analytics and Planning (RAP) suite of applications can be extended and customized to fit the needs of your implementation.

Custom applications, services and interfaces can be developed for AI Foundation using the Innovation Workbench module. Innovation Workbench is also the first choice for programmatic extensibility within RAP applications and provides access to data from both PDS and AIF.

Planning application configurations can be extended using the native RPASCE platform functionality, and further extended using Innovation Workbench.

Retail Insights can be extended with custom datasets brought into the application using Data Visualizer. This chapter will provide an overview of the RAP extensibility capabilities with links and references to find more information.

> **Note:**
>
> Before continuing with this section, please read the application-specific implementation/user guides.

This chapter includes the following sections:

AI Foundation Extensibility

The Innovation Workbench as a part of the AI Foundation module consists primarily of Application Express (APEX) and Data Studio. These tools provide significant extensibility features for custom analytical applications, advanced data science processes, 3rd party integrations, and much more. Some examples of IW capabilities for AI Foundation include:

- Custom database schema with full read/write access allows you to store data, run queries, perform custom calculations, and debug integrations across the RAP platform
- Use advanced Oracle database features like Oracle Data Mining (ODM) and other machine-learning models
- Use Notebooks in Data Studio to create custom Python scripts for analytics, data mining, or machine learning
- Notebooks and APEX jobs can be scheduled to run automatically to refresh data and calculations
- Create Restful API services both to request data from IW out to other systems and to consume non-Oracle data into the platform

- Build flat file integrations into and out of IW for large data movements and custom dataset extensions
- Build custom monitoring and utilities to manage integrations and science models with business IT processes

More details on Innovation Workbench features and examples of custom extensions can be found in the *AI Foundation Implementation Guide* chapter on Innovation Workbench.

Custom Hooks for IW Extensions

The AIF DATA POM schedule contains 20 generic jobs that support execution of custom PL/SQL procedures in IW. Use these jobs in POM to hook your extensions directly to the AIF DATA batch schedule for automated nightly execution without needing to rely on `DBMS_SCHEDULER` jobs. 10 of the jobs (those with `PRELOAD` in the name) are placed just after the staging table loads, but before any internal tables are populated, allowing you to edit or create data in staging tables directly. The other 10 jobs are placed at the end of the AIF DATA batch for custom post-load activities.

Table 9-1 AIF DATA Jobs for IW

Job Name	Description
RI_IW_CUSTOM_1_JOB	Execute Custom IW Procedure 1
RI_IW_CUSTOM_2_JOB	Execute Custom IW Procedure 2
RI_IW_CUSTOM_3_JOB	Execute Custom IW Procedure 3
RI_IW_CUSTOM_4_JOB	Execute Custom IW Procedure 4
RI_IW_CUSTOM_5_JOB	Execute Custom IW Procedure 5
RI_IW_CUSTOM_6_JOB	Execute Custom IW Procedure 6
RI_IW_CUSTOM_7_JOB	Execute Custom IW Procedure 7
RI_IW_CUSTOM_8_JOB	Execute Custom IW Procedure 8
RI_IW_CUSTOM_9_JOB	Execute Custom IW Procedure 9
RI_IW_CUSTOM_10_JOB	Execute Custom IW Procedure 10
RI_IW_PRELOAD_CUSTOM_1_JOB	Execute Custom IW Preload Procedure 1
RI_IW_PRELOAD_CUSTOM_2_JOB	Execute Custom IW Preload Procedure 2
RI_IW_PRELOAD_CUSTOM_3_JOB	Execute Custom IW Preload Procedure 3
RI_IW_PRELOAD_CUSTOM_4_JOB	Execute Custom IW Preload Procedure 4
RI_IW_PRELOAD_CUSTOM_5_JOB	Execute Custom IW Preload Procedure 5
RI_IW_PRELOAD_CUSTOM_6_JOB	Execute Custom IW Preload Procedure 6
RI_IW_PRELOAD_CUSTOM_7_JOB	Execute Custom IW Preload Procedure 7
RI_IW_PRELOAD_CUSTOM_8_JOB	Execute Custom IW Preload Procedure 8
RI_IW_PRELOAD_CUSTOM_9_JOB	Execute Custom IW Preload Procedure 9
RI_IW_PRELOAD_CUSTOM_10_JOB	Execute Custom IW Preload Procedure 10

Here are the steps to enable the functionality:

1. Create a package and package body for the process that needs to be run as a custom job. After logging into AI Foundation, from the task menu go to **Innovation Workbench -> Manage Workbench** and then in APEX go to **SQL Workshop**. On the right-hand side click **Package** under **Create Object** and proceed with creating the package specification, package body, and create the procedure inside the package body.

2. Back in AI Foundation, use the Manage System Configuration screen in the Control Center to modify the table `RI_CUSTOM_JOB_CFG` and edit values for the following columns:
   a. `PACKAGE_NAME`: Enter the name of the package that was created in IW.
   b. `PROCEDURE_NAME`: Enter the name of the procedure in your package that was created in IW.
   c. `PROCEDURE_DESCR`: Enter a description, if desired.
   d. `RUN_TIME_LIMIT`: The run time limit is `900` seconds by default. It can be changed to a different value if needed. If the custom process runs for longer than the value indicated in `RUN_TIME_LIMIT` when running as a part of the batch process, the custom process will stop and move on to the next job/process.
   e. `CONNECTION_TYPE`: Valid values are `LOW` and `MEDIUM`. This value should almost always be `LOW` unless the job is supposed to run a process that would need multiple threads. `HIGH` is not a valid value. If `HIGH` is entered, it will switch to `LOW` by default when the job runs.
   f. `ENABLE_FLG`: Set this value to `Y` to indicate that this job should be executed as part of the batch process.
3. The POM jobs should be enabled in the Nightly batch once configured. Alternatively, you may use the ad hoc process `RI_IW_CUSTOM_ADHOC` to run the jobs outside of the batch.

Because these jobs are added as part of the nightly batch, they do not allow extended execution times (>900 seconds) by default. If you are building an extension that requires long-running jobs, those should be scheduled using the `DBMS_SCHEDULER` package from within IW itself so that you don't cause batch delays.

Planning Applications Extensibility

Planning applications have a number of paths for extensibility. The configuration itself is extensively customizable in terms of business logic and rules, fact definition, data integration, and the user interface.

Provided that the application extensibility framework is followed, any customizations will be preserved in future patch and service upgrades.

Additionally, there is programmatic extensibility provided through the incorporation of PDS into Innovation Workbench. This allows the customer to extend the features provided by planning applications and build their own novel functionality. In addition, they can leverage the rich abilities and data already provided by IW.

Supported Application Configuration Customization

The following sections list the customizations that are allowed for application configuration. The following configuration components can be customized:

> **Note:**
>
> These customizations must be made through RPASCE Configuration Tools.

- Solution
- Measures

- Rules and Rule groups
- Workbooks and worksheets
- Hierarchy
- Taskflow
- Styles

For the customizations to be recognized, all names of custom-realized measures, rule sets, rule groups, rules, workbooks, worksheets, and styles should begin with the prefix `c_` or `C_`.

Custom worksheets may only be added into existing workbook tabs for plug-in generated solutions.

Rules for Customizing Hierarchy

The following hierarchy customizations can be made to the application configurations:

- New hierarchies may be added, and/or new dimensions added to the existing hierarchy. However, no dimension may be added to a calendar hierarchy that is lower than “day”. Finally, no change can be made to internal hierarchies.
- Changes are permitted to the labeling of existing hierarchies or dimensions.
- All dimension and roll-up orders in the product, RHS product, location, and RHS location hierarchy must be preserved in the custom configuration.

Rules for Adding Measures

The following rules apply when adding measures to the application configurations:

- Customers may add new custom measures into the custom solution and reference them as an external measure in the extensible solutions.
- Customers can also add new custom metrics as a major component in the extensible solutions. It is strongly recommended to not mix custom metrics with application metrics.
- Custom measures should follow the naming convention and should begin with a `C_` or `c_` prefix.
- Only the published GA measures can be used in custom rules and custom workbooks. Only writable GA measures can be used on the left-hand side of a rule expression. The read-only GA measures can only be used on the right-hand side of the rule expression.

Publishing Measures

The published GA measures can be divided into the following categories:

Read only—can only be used on the right-hand side of the expression

Writable—can be used on both the left-hand side and right-hand side of the expression

RuleGroupOnlyWritable—a specific measure that can be read/written in the specified rule group

Loadable—measures that can be loaded using OAT and can be present in the custom load batch control file

WorkbookMeasureOverride—measures which property can be overridden in the associated workbook

ReadableExecutionSet—list of GA batch control execution set names that can be called from within a custom batch control execution file

The list of published measures will change based upon configuration. Therefore, the list is dynamically generated at each configuration regeneration.

The contents of the list are saved in a file named: `publishedMeasures.properties`.

The file is located under `[config]/plugins`. Before writing custom rules, regenerate your application configuration and then open the file to search for published application measures.

Custom Measure Characteristics:

- Each line of the file contains multiple fields that are pipe ("**|**") separated.
- The first field is one of the category names previously listed.
- The second field is the name of the measure or execution set.
- The third field is the measure label.
- For `RuleGroupOnlyWritable`, the fourth field is the rule group name.
- For `WorkbookMeasureOverride`, the fourth field is the name of the workbook in which this measure is allowed to be overridden.

Sample Custom Measure

```
ReadOnly|PreSeaProf|Seasonal Profile
ReadOnly|activefcstitem01|Active Forecast Items
ReadOnly|activefcstitem07|Active Forecast Items
```

Generally, forecasting parameter overrides such as Forecast Method Override, Custom Exception, Custom Exception Metric, auxiliary inputs to applications such as Promotion Aggregation Profile, and Grouping Membership are writable because an implementer may set them up through customized rules.

Rules for Adding Custom Rules

The following rules apply when adding custom rules to the Application configuration:

- Custom rule sets, rule groups and rule names should begin with the `C_` or `c_` prefix.
- Custom rule groups should not include any GA rules.
- Custom rules can use the published read-only GA measures listed in the `publishedMeasures.properties` file. However, the custom rules cannot modify the value of the read-only GA measure. Hence the read-only GA measure cannot appear on the LHS of a custom rule.
- Custom Rules can be added to custom rule group. They can also be added to the plug-in generated GA workbook rule groups such as load rule group, calc rule group, refresh rule group, commit rule group and custom menu rule. However, Custom Rules cannot be added to a plug-in generated batch rule group.

Rules for Workbooks and Worksheets Extensibility

The following rules apply when adding custom rules to the workbooks and worksheets extensibility:

- New Custom workbook and worksheets names should begin with the `C_` or `c_` prefix.

- Apart from the Custom Solution, custom workbooks can also be added to the extensible GA solutions.

Workbook Measure Override Extensibility

Certain GA measures can be overridden in the GA workbook. These measures are listed in the `WorkbookMeasureOverride` section of the `published<app>Measures.properties` file.

For example:

```
WorkbookMeasureOverride|<measure name>|<measure label>|<workbook template>
```

This indicates that the measure can be overridden in the workbook.

The following rules apply to override measure properties:

- Base State and Agg State can be overridden.
- Range property of static picklists can be overridden.

> **Note:**
>
> Options can only be removed; new options cannot be added.

Elapsed Lock Override

Elapsed Lock Override on RPASCE is supported in the following scenarios:

- Custom measures in a workbook can have the Elapsed Lock Override set to `true`.
- Custom workbooks can have this field set to `true` for GA measures that are in the Writable list of the published measures.

> **Note:**
>
> If a GA measure has not been enabled as Elapsed Lock Override, the following steps can achieve the same behavior:
>
> 1. Make sure the GA measure is writable.
> 2. Register a custom measure and load it from the GA measure.
> 3. Set the custom measure as Elapsed Lock Override.
> 4. Edit the custom measure in the workbook.
> 5. Commit the custom measure back into the GA measure.

Rules for Adding Custom Real-time Alerts into Existing Workbooks

Perform the following steps when adding custom real-time alerts into existing workbooks.

> **Note:**
>
> These steps must be performed using RPASCE Configuration Tools. Copying, pasting or direct editing of XML files is unsupported.

1. To add custom real-time alert into existing workbooks, all measures related to the custom real-time alert need to be added to the workbook.
2. Create a style for the custom real-time alert in the configuration.
3. Create a custom real-time alert in a workbook using the measures and style created from the previous steps.
4. If a real-time alert defined in custom solution will be used in a GA workbook, the real-time alert measure should be imported as an external measure in the corresponding GA solution.
5. We must ensure that the rule group consistency is maintained while adding any custom rules that might be needed to calculate an alert measure.

The application plug-in will preserve a custom real-time alert during regeneration

Adding a Custom Solution

A custom solution is a separate solution within the configuration. It can be used to accommodate custom workbooks, rules, alerts to do custom reporting, custom logic, and threshold alerts by using GA measures (based on the extensible GA measures in Table 9-2). In addition, measures and alerts defined in the custom solution can be plugged into existing workbooks in GA solution based on the contexts defined. Clients are allowed to create their own custom solutions by following the rules mentioned above. To use a GA measure in custom workbooks, the GA measure should be imported as an external measure in custom solution.

Adding Custom Styles

New styles can be added in the Style Definition window of Configuration Tools. The custom style name should be prefixed with either `c_` or `C_`. Style names that do not adhere to the naming convention will be caught during the configuration validation. Any new style added will be retained during upgrades and patches.

Validating the Customized Configuration

A script, `ra_config_validation.sh`, has been provided to allow the customer or implementer to validate that the customizations conform to the rules outlined above. For details of the script, refer to Configuration Validation.

This script can be run on Windows with the application starter kit. To do this, the implementer will need to make sure that they have a pristine copy of the GA configuration as well as the custom configuration.

For example, if the GA configuration has been copied to `C:\Oracle\configurations\GA\RDF` and the custom configuration is in `C:\Oracle\configurations\RDF`, then the script can be called from a Cygwin zsh shell:

```
$RPAS_HOME/bin/ra_config_validation.sh -n RDF -d /cygdrive/c/Oracle/
configurations -c /cygdrive/c/Oracle/configurations/GA/RDF/RDF.xml
```

Successful Run of the Validation Script

If all the validations pass, it will output the following message:

Message for Successful Run of Validation Script

```
09:04:47 : INFORMATION : ra_config_validation.sh[0] - ra_config_validation.sh
completed.
09:04:47 : INFORMATION : ra_config_validation.sh[0] - Program completed
successfully.
09:04:47 : INFORMATION : ra_config_validation.sh[0] - Exiting script with
code: 0
```

Unsuccessful Run of the Validation Script

If all the validations do not pass, it will output the following message:

> **Note:**
>
> The bold line shows where the details of the validation failure are in the log. (In the actual log, this line is not bold.)

Message for Unsuccessful Run of Validation Script

```
09:15:12 : INFORMATION : ra_config_validation.sh[0] - For details of
validation, look in '/cygdrive/d/retek/logs/2017-07-18/
rdf_config_validation.091506.1/rdf_config_validation.log'.
09:15:12 : INFORMATION : ra_config_validation.sh[0] - _call executing command
'execplug-inTask.sh RDF:com.retek.labs.rdf.plug-
in.installer.RDFConfigurationValidation /cygdrive/c/Oracle/
configurations/GA/RDF/RDF.xml /cygdrive/c/Oracle/configurations RDF'
09:15:17 : INFORMATION : ra_config_validation.sh[0] - _call of command
'execplug-inTask.sh RDF:com.retek.labs.rdf.plug-
in.installer.RDFConfigurationValidation /cygdrive/c/Oracle/
configurations/GA/RDF/RDF.xml /cygdrive/c/Oracle/configurations RDF' complete
09:15:17 : ERROR : ra_config_validation.sh[0] - Nonzero exit status code.
09:15:17 : INFORMATION : ra_config_validation.sh[0] - Exiting script with
code: 9
```

Taskflow Extensibility

The application taskflow is extensible, the implementer can add custom taskflow components such as activities, tasks, steps, tabs, and worksheets. Any custom taskflow component added to a GA taskflow component will be retained after plug-in automation. As part of extensibility, applications provide a mechanism wherein the implementor can hide certain components of the GA configuration and taskflow by editing a property file. The property file is a simple text file named `extend_app.properties` and is located inside the plug-in directory of the configuration. A sample file is included in the plug-ins directory of the GA configuration for reference.

For example, `<App>\plug-ins\extend_app.properties`

The format of the file is shown as:

```
Stage|Component|Action|Value
```

An example entry is:

```
Customization | Worksheet  | Hide | activity_ni.task_niattmaint.NITREVSht1
```

Each line consists of four fields separated by the `|` character. The value field can contain a comma-separated list of values. Note that the value field should specify the fully qualified name of the taskflow component. Refer to the sample file. Any line that begins with a `#` character is considered a comment line and is ignored.

The names of the Taskflow entities can be found in the `taskflow.xml` file located in the configuration directory.

The various GA configuration components that can be hidden are listed in the following table:

Component	Description
Activity	Hides the specified taskflow activity. The value field is the taskflow activity name.
Task	Hides the specified Taskflow task. The value field is the taskflow task name.
Step	Hides the specified Taskflow step. The value field is the taskflow step name.
Tab	Hides the specified Taskflow tab. The value field is the taskflow tab name.
Worksheet	Hides the specified worksheet. The value field is the worksheet name.
Realtime Alert	Hides the specified Real-time Alert. The value field is the real-time alert name.

Customizing the Batch Process

This section describes how to customize the GA batch process to meet the business needs of the retailer. Details on the GA batch process are described in the *Oracle® Retail Inventory Planning Optimization Cloud Service-Demand Forecasting / Inventory Planning Optimization Cloud Service-Lifecycle Allocation and Replenishment Administration Guide*. The Configured Batch tasks have the following tasks related to batch control:

- Retrieve Batch Control File – allows the current batch control files to be retrieved for inspection and modification.
- Update Batch Control File – After inspecting the current batch control files, the implementor can edit the batch control files to customize the batch process.

Details on the previous two tasks are described in the *Oracle Retail Predictive Application Server Cloud Edition Implementation Guide*.

The application batch process is based on the RPASCE Enterprise Edition Batch Framework, which makes use of a set of control files. Table 9-2 lists the batch control files that can be customized. For detailed information on the RPASCE Batch Framework, refer to the *Oracle Retail Predictive Application Server Implementation Guide*.

Table 9-2 Customizable Batch Control Files

Control File	Description
batch_exec_list.txt	This is the controller and entry point for all the other services, specifying groups of services to be run in a specific order.
batch_calc_list.txt	This control file groups all the calc services that need to run using mace.
batch_refresh_list.txt	This control file groups all Workbook refresh rule groups
batch_loadmeas_list.txt	This control file groups measures that need to be loaded into domain using the measure load service
batch_exportmeas_list.txt	This control file groups measures that need to be exported out of the domain using export measure service.
batch_xform_list.txt	This control file handles the transform file service to perform file transformations to support simple integration capabilities.
batch_oat_list.txt	This file lists the configured batch tasks that appear in the OAT drop down list.

Custom Hooks and Boolean Scalar Measures for Flow Control

There are two ways to customize the batch control files:

Custom Hooks

Boolean Scalar Measures for Flow Control

The custom hooks are an optional batch set executed by GA batch control files. The implementer can define the contents of these batch sets in the customized batch control files that can be uploaded. If these hooks are not defined, then the batch process skips these hooks; otherwise, its contents are executed.

The application also defines a list of Boolean Scalar Measures in the domain to control whether certain GA-defined batch sets can be skipped or not. The following tables list the hooks and Boolean Scalar Measures.

Please refer to Application Specific Batch Control Information for details and examples.

Batch Control File Customization Guidelines

Follow these guidelines for Batch Control File Customization:

- The file `batch_oat_list.txt` is the only batch control file in which customers can overwrite the label for GA set names listed in OAT.
- For all other batch control files, avoid overwriting GA set names. GA batch control files have provided various hooks for the batch process. For additional custom steps, try to put them into the hooks.
- GA batch control files have provided a mechanism to skip certain GA steps using boolean scalar measure that can be set in the domain.
- For a GA hierarchy that is unused in your implementation (such as attribute hierarchy), provide an empty hierarchy file. For unused GA measures, there is no requirement to provide the data file. RPASCE will be able to skip it if no files are provided.
- Do not remove any GA `clnd` hierarchy reorder steps; these steps are essential to the proper functioning of the application.

- For ease of maintenance, all custom batch set name or step names should be prefixed with `c_`

Examples

The following is an example of custom `batch_exec_list.txt`, `batch_calc_list.txt`, `batch_loadmeas_list.txt`, and `batch_exportmeas_list.txt`.

In this example, the following modification were added to the batch `_weekly` process:

- Hierarchy and measure data file were unpacked.
- Custom measures were loaded after GA measure load.
- Outlier indicators for preprocessing were calculated use custom rules
- Custom approval alerts were run after GA alerts and before approval
- Promotion effects were exported after GA exports

Custom Batch Control Validation

The extensible / custom batch control files need to follow the guidelines previously to futureproof the retailer. That means the retailer should receive software updates without breaking the existing customizations. To ensure that the batch control file guidelines are adhered to, a batch control validation module has been added.

The `ra_config_validation.sh` script has an optional parameter `-b <parent directory of batch control files>` which will validate the batch control files.

Batch control validation rules:

- Apart from `batch_oat_list.txt`, none of the set names in the other batch control files can be overridden. That is, GA set names cannot be used in custom batch control files.
- None of the custom batch control files can call the GA set names.
- The `batch_calc_list.txt` file can only specify custom rule group names. Cannot specify expressions and GA rule group names.
- The `batch_loadmeas.txt_list` file can specify measures that are listed in the Loadable or Writable list of the published measures in the `published<app>Measures.properties` file
- The `batch_exportmeas_list.txt` file can specify measures that are listed in the ReadOnly or Writable list of the published measures in the `published<app>Measures.properties` file.
- All custom set names should have a prefix of `c_`.

> **Note:**
>
> The batch control validation is called automatically during domain build or patch. It is also called when the batch control files are uploaded using the Upload Batch Control files from OAT.

Dashboard Extensibility

Currently, IPOCS-Demand Forecasting supports Dashboard Extensibility by allowing the Dashboard Settings configuration file to be customized. The other planning applications, such as MFP and AP, support customizing the dashboard, but these are not extensible (please refer to Customizing the MFP/AP Dashboard).

IPOCS-Demand Forecasting Dashboard Extensibility

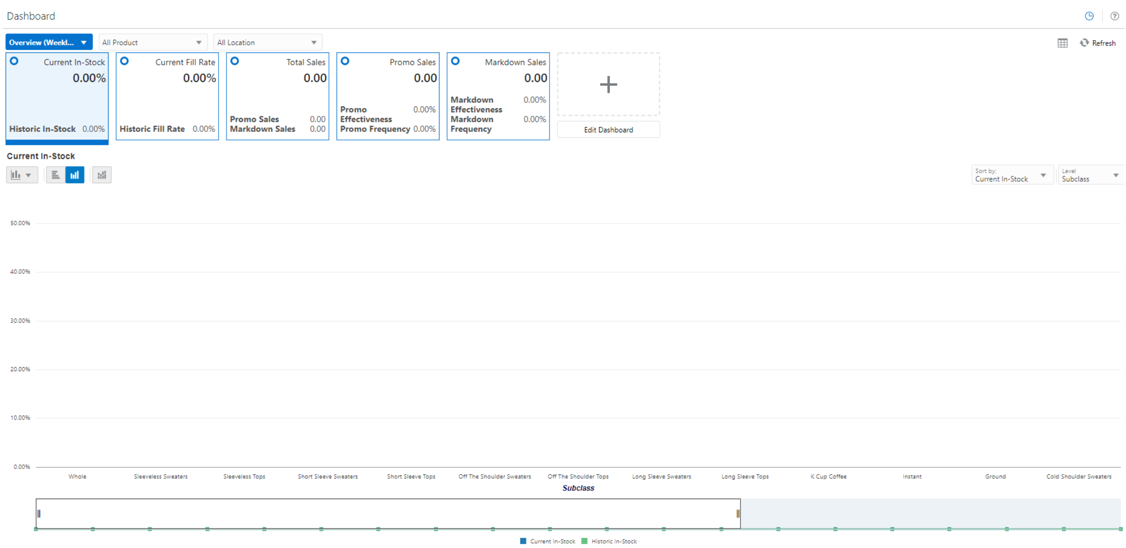
For detailed information on Dashboard components, please refer to the chapter, “Configuring Dashboards in RPASCE EE” in the *Oracle Retail Predictive Application Server Cloud Edition (RPASCE) Configuration Tools User Guide*.

As part of extensible dashboard, the following are supported:

- Adding custom Metric and Exception profiles.
- Adding a custom tile to GA Metric and Exception profiles.
- Removing GA tiles and profiles.

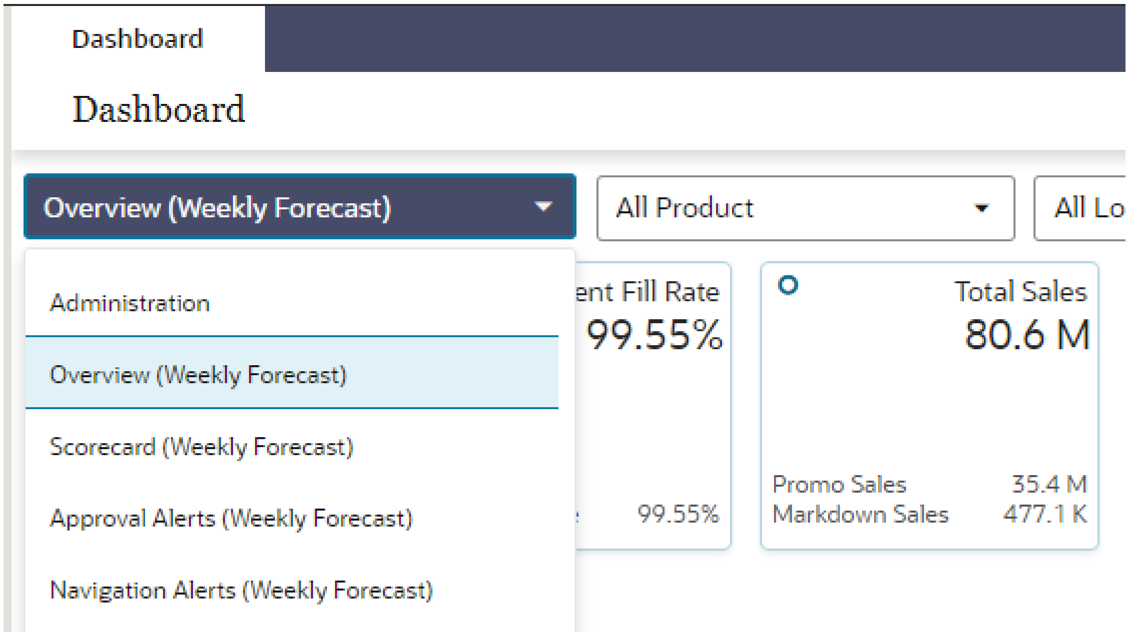
Figure 9-1 shows the IPOCS-Demand Forecasting Dashboard as seen in the UI. It consists of two Metric profiles and two Exception profiles.

Figure 9-1 IPOCS-Demand Forecasting Dashboard



In Figure 9-2,the Overview Metric profile is selected, and the Total Sales tile is highlighted with two sub-measures: Promo Sales and Markdown Sales.

Figure 9-2 IPOCS-Demand Forecasting Dashboard Selection

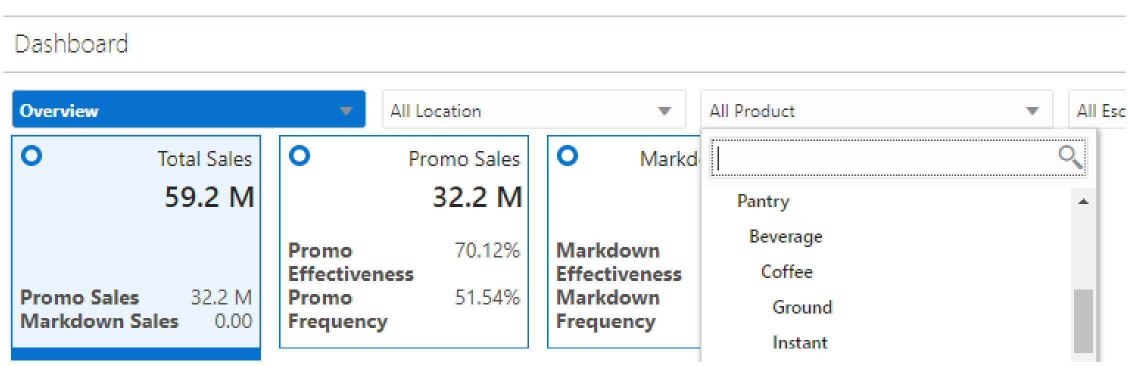


> **Note:**
>
> The Exception profiles consist of Exception Tiles, and the Metric Profile consists of metric tiles of the type Comparison Tile. Currently, IPOCS-Demand Forecasting does not support the Variance Metric tile.

Dashboard Intersection

The IPOCS-Demand Forecasting GA Dashboard workbook is built at the Sub-class, District level which is controlled by the Dashboard Intersection specified in the IPOCS-Demand Forecasting plug-in. Refer to the "IPOCS-Demand Forecasting / IPOCS-Lifecycle Allocation and Replenishment Configuration" section in the *Oracle® Retail Inventory Planning Optimization Cloud Service-Demand Forecasting/ Inventory Planning Optimization Cloud Service-Lifecycle Allocation and Replenishment Implementation Guide*. The Dashboard intersection also defines the level to which we can drill down the Product and Location filters in the Dashboard.

Figure 9-3 Product / Location Filters in the Dashboard



Process to Customize the Dashboard

Dashboard profiles correspond to a worksheet in the Dashboard workbook template in the configuration; and the measures displayed in the tiles are measures present in the worksheet corresponding to that profile. Customizing the dashboard is a three-step process:

In the Configuration, add the worksheet, measures, and rules to the Dashboard workbook template.

Regenerate the configuration by running the plug-in automation and then validate the configuration by running the `ra_config_validation.sh` script. Refer to the section, Validating the Customized Configuration for more information.

Customize the GA Dashboard Settings file in the Deployment Tool.

> **Note:**
>
> The Deployment Tool is a utility within the Configuration Tools. Refer to the section, *Deployment Tool – Dashboard Settings Resource in the Oracle Retail Predictive Application Server Cloud Edition (RPASCE) Configuration Tools User Guide*.

The IPOCS-Demand Forecasting GA Dashboard Settings configuration file is found within the configuration: `RDF\plugins\dashboardSettings.json`

Steps to add a custom profile:

1. In the Configuration Tool, add custom worksheet and measures to the worksheet in the dashboard workbook template in the configuration. Also add load/calc rules for the measures.
2. In the Deployment Tool, open the GA Dashboard Settings configuration file.
3. Add the custom profile (Exception or Metric) to the Dashboard Settings configuration file.
4. Save the file in the Deployment Tool.

Steps to add a custom tile:

1. Identify the profile and worksheet to which the custom tiles need to be added.
2. In the Configuration Tool, add the custom measures to the corresponding worksheet. Also add load/calc rules for the measures.
3. In the Deployment Tool, open the GA Dashboard Settings configuration file.
4. Based on whether Exception or Metric profile, add the Exception tile or Comparison Metric Tile.
5. Save the file in the Deployment Tool.

Steps to remove GA tiles and profiles:

> **Note:**
>
> Do not remove the GA measures or worksheet from the Dashboard workbook template in the configuration.

1. In the Deployment Tool, open the GA Dashboard Settings configuration file.
2. Delete the GA profile or tile.
3. Save the file in the Deployment Tool.

Save the Dashboard Settings Configuration file in the same location in the configuration, that is: `RDF\plugins\dashboardSettings.json`. Because this file is stored inside the configuration, whenever the customer uploads the configuration to the **Object Store**, the customized Dashboard Configuration file will be used by the application during the domain build or patch process.

Once the domain is built or patched, if minor changes need to be made to the Dashboard that do not require a configuration change, then RPASCE provides a mechanism to Upload and Retrieve JSON files from the application.

This is supported through the **Configured Batch OAT task -> Manage JSON File** option. Refer to the *Oracle Retail Predictive Application Server Cloud Edition (RPASCE) Administration Guide* for detailed information on the OAT tasks.

Steps to Retrieve/Upload the Dashboard Configuration File:

1. Go to the **Configured Batch OAT task -> Manage JSON Files -> Retrieve** option.
2. The dashboard settings file will be downloaded into the **Object Store** as `RDF_json.tar.gz`
3. Un-tar the file and open it in the Deployment Tools.
4. Edit the file. Note that only minor updates that do not require a configuration change can be made at this time.
5. Save the file and zip it up as `RDF_json.tar.gz` and then upload it to the **Object Store**
6. Then go to the **Configured Batch OAT task -> Manage JSON Files -> Upload option**.
7. Log out and log in to the client.
8. The Dashboard should be updated with the changes

Applying Changes to the Cloud Environment

To implement these changes in the cloud environment, it is necessary to either build a new domain or patch the domain. Refer to the "Build/Patch Application" chapter in the *Oracle® Retail Inventory Planning Optimization Cloud Service-Demand Forecasting / Inventory Planning Optimization Cloud Service-Lifecycle Allocation and Replenishment Administration Guide*.

Customizing the MFP/AP Dashboard

The application Dashboard gets data from the regular dashboard workbook template like any other workbook segments to define the measures used in metric tiles that are shown in the dashboard. This Dashboard can be customized using the same extensibility rules for regular workbooks for adding new measures into that dashboard workbook (`pl_db`). The customer can then update the MFP/AP Dashboard JSON file to include the newly added custom measures to show as tiles in the Dashboard.

Following are the steps for customizing the Dashboard:

1. Update application Configuration to include the required new custom measures and rules to include those measures in the existing dashboard template (`pl_db`) in the application Configuration within regular extensibility framework. Patch the domain with the new updated configuration.

2. Download the application dashboard JSON file (`dashboardSettings.json`) from the Starter kit or directly from the customer-provisioned environment by running the Online Administration Tools task **Patch Application Task -> Manage JSON Files -> Retrieve JSON files to Object Storage**. This will download the JSON file into the Object Storage location at `outgoing/dashboardSettings.json`.
3. Open the downloaded dashboard JSON file using the RPASCE **Configuration Tools -> Utilities -> Deployment Tool** and selecting the **Open** option under `dashdoardSettings.json`.
4. It should open the dashboard JSON file in edit mode. The customer can then edit the dashboard to add the newly added measures into their required profiles. They can also add new profiles or change profiles but can only use the measures available in the dashboard workbook. For more information on working with the JSON file using RPASCE Configuration Tools, see the *Oracle Retail Predictive Application Server Cloud Edition Configuration Tools User Guide*.
5. Once the JSON file is updated, it can be uploaded into the MFP environment by uploading the file to the Object Storage location as `incoming/config/dashboardSettings.json`, and running the Online Administration Tool task **Patch Application Task -> Manage JSON Files > Update JSON files from Object Storage**. Successful completion of the task will copy the file to the required location under the application domain.
6. After uploading, rebuild the dashboard to view the updated dashboard.
7. The entire process can be validated in the Virtual machine before trying to upload the completed JSON file into the customer environment.

RAP Integration Interface Extensibility

`interface.cfg` is a PDS configuration file that defines the bidirectional exchange of hierarchy and fact data between AIF and PDS, through RDX. For detailed information about the interface configuration file, please refer to the *Oracle Retail Predictive Application Server Cloud Edition Implementation Guide*.

The extensibility of `interface.cfg` is supported provided that the below guidelines are followed.

> **Note:**
>
> The permissible and restricted interface customization is published in the file `publishedMeasures.properties` located in the `[config]/plugins` directory.

Type	Rule	Comments/Sample Entries
Hierarchy	A new hierarchy interface can be defined provided the table already exists in RDX.	
Fact	A new fact interface can be defined provided the table already exists in RDX.	Note that the interface parameter (second field) must be different than the GA interface.

Type	Rule	Comments/Sample Entries
Views	Custom views cannot be defined. GA interfaces that are views can be customized.	
Hierarchy	The dimension mapping for a GA hierarchy interface can be modified, provided it is allowed in the published property file.	`RSE_FCST_DEMAND_DTL_CAL_EXP:L01:DATA:rdf_sysbaseline01:DEMAND_FCST_QTY::custom_modRSE_FCST_DEMAND_DTL_CAL_EXP:L01:DATA:rdf_sysfrcst01:BASELINE_FCST_QTY::custom_mod`
Hierarchy	Custom dimensions can be added to a GA interface and mapped to existing RDX fields.	For example, if adding custom dimension 'area' to the `LOC` hierarchy, customize the `W_PDS_ORGANIZATION_D` interface: `W_PDS_ORGANIZATION_D:PDS:HDM50:AREA:AREA::custom_add` Note that custom dimensions should have IDs >= 50
Hierarchy	Custom (New) hierarchy can be added, provided table exists in RDX..	Note that custom dimensions should have IDs >= 50

Facts (Import)	Existing GA facts can be imported from a different source field (controlled by an extensibility property file).	GA entry:
		`RSE_FCST_DEMAND_DTL_CAL_EXP:L_CF_:DATA:rdf_sysbaseline_CF_:BASELINE_FCST_QTY:`
		Custom entry:
		`RSE_FCST_DEMAND_DTL_CAL_EXP:L_CF_:DATA:rdf_sysbaseline_CF_:FLEX_FIELD1: :custom_mod`

Facts (Import)	Custom facts can import from existing fields from a source table in RDX.	`RSE_FCST_DEMAND_SRC_EXP:L01:DATA:c_outageind01:STOCKOUT_IND::custom_add`
Facts (Export)	GA facts can be exported to another external field alongside the GA entry.	`RDF_APPR_FCST_CAL_EXP:APPC01:DATA:rdf_appbaseline01:APPR_BASELINE_FCST:RDF_APPR_FCST_CAL_EXP:APPC01:DATA:rdf_appbaseline01:APPR_BASELINE_FCST1::custom_add`
Facts (Export)	The external field can be populated by a different fact. Delete the GA entry and add the custom entry.	`RDF_APPR_FCST_CAL_EXP:APPC01:DATA:rdf_appfrcst01:APPR_DEMAND_FCST::custom_delRDF_APPR_FCST_CAL_EXP:APPC01:DATA:c_appfrcst01:APPR_DEMAND_FCST::custom_add`
Facts (default value)	The default value can be customized for only facts allowed in property file.	

Type	Rule	Comments/Sample Entries
InterfaceFilters (Allow)	Filters for an interface can be customized, provided it is allowed and not restricted in the property file. Note that the GA filter entry needs to be marked as deleted and then a custom entry added.	`Validation | InterfaceFilter | Allow | RSE_FCST_DEMAND_DTL_CAL_EXP,RSE_FCST_DEMAND_DTL_EXP` means the above interfaces can have the filters customized.
InterfaceFilters (Restrict)	Although an interface is allowed for filters to be customized, some filters can be marked as restricted and cannot be customized.	`Validation | InterfaceFilter | Restrict | RSE_FCST_DEMAND_DTL_CAL_EXP:RUN_ID;BASELINE_FCST_QTY,RSE_FCST_DEMAND_DTL_EXP:RUN_ID` In this case, the `RUN_ID` and `BASELINE_FCST_QTY` filters cannot be customized. Other filters in this interface can be customized.

> **Note:**
>
> 1. Any customization in the `interface.cfg` file should be marked with a `custom_???` keyword in the 7th field.
> 2. Note the keywords used to extend the GA `interface.cfg`:
>   a. `custom_mod`: to indicate an existing GA entry is being modified
>   b. `custom_del`: to indicate an existing GA entry is being removed
>   c. `custom_add`: to indicate a custom entry is being added.
> 3. If we are deleting and adding an entry, make sure they are consecutive entries.

Validations for a custom `interface.cfg` file:

- Custom entries cannot have more than seven (7) fields.
- Filter entries can only have `custom_add` or `custom_del` keywords.
- Entries cannot be deleted from the Hierarchy interface. Therefore, `custom_del` entries are not valid for the Hierarchy interface.
- Dimensions specified in the Hierarchy interface must be valid dimensions in the configuration.
- Custom dimensions added to the Hierarchy interface should have a dimension ID greater than or equal to 50.
- For Fact/Data interfaces, the dimensions/IDs cannot be modified.
- Only Hierarchy dimensions published in the property file can be modified.
- Only Interface and Facts published in the property file can be modified.

- Only Interface Filters published and not restricted in the property file can be edited.

Follow this process to update the `interface.cfg` file:

1. Download the Application interface configuration from OAT
2. Update the `interface.cfg` using the previously listed guidelines.
3. Upload the updated `interface.cfg` to object store and then patch or build the application.

Application Specific Batch Control Information

Below sections describes Batch Control details that are specific to IPOCS-Demand Forecasting:

Table 9-3 Custom Hooks

Hook	Description
hook_calc_attb_CF_	This hook is executed right after GA attributes exception `navifin_CF_` is calculated and before approval business rule group are calculated. If any custom calculated attributes have been set up to be used in approval by implementor. This is the place to insert custom attributes calculations. `_CF_` needs to be replaced by a level number.
hook_frcst_adjust_CF_	This hook is provided to add custom forecast adjustment calculations. This hook is before the business rule group related calculation, approval, and navigation logic. `_CF_` needs to be replaced by a level number.
hook_frcst_alert_CF_	This hook is provided to merge the user specified parameters associated with approval business rule group before running exceptions. After merging the user specified parameters, the custom approval exceptions and exception metric should be executed. `_CF_` needs to be replaced by a level number.
hook_frcst_approval_CF_	This hook is provided to perform any post-processing to approval forecast after GA approval step. `_CF_` needs to be replaced by a level number.
hook_navi_attb_CF_	This hook is provided so that implementor can calculate the custom calculated attributes used in the navigation business rule groups. `_CF_` needs to be replaced by a level number.
hook_populate_aprvrulg_eligiblemask_CF_	This hook is for populate rulgeligmask_CF measure using custom logic. This measure is the eligible mask at sku/store/rulegroup. It can be populated with custom logic to calculate eligible items for approval business rule groups. `_CF_` needs to be replaced by a level number.
hook_post_export	This hook is after export.
hook_post_forecast	This hook is between forecast and export.
hook_post_preprocess	This hook is after the preprocessing phase and before generating the forecasts.

Table 9-3 (Cont.) Custom Hooks

Hook	Description
hook_pre_forecast	This hook is after New Item calculation and before the forecast generation step.
hook_pre_post_data_load	This hook is between GA measure load and `post_data_load` rule group run.
hook_IPO_COM_DATA_IMP_OBS_D hook_IPO_COM_DATA_IMP_OBS_W hook_IPO_COM_DATA_IMP_RDX_D hook_IPO_COM_DATA_IMP_RDX_W	This hook is for the calling steps using any import of common data interfaces.
hook_IPO_COM_HIER_IMP_OBS_D hook_IPO_COM_HIER_IMP_OBS_W hook_IPO_COM_HIER_IMP_RDX_D hook_IPO_COM_HIER_IMP_RDX_W	This hook is for the calling steps using any import of common hierarchies.
hook_IPO_HIER_IMP_OBS_D hook_IPO_HIER_IMP_OBS_W hook_IPO_HIER_IMP_RDX_D hook_IPO_HIER_IMP_RDX_W	This hook is for the calling steps using any import of application-specific hierarchies.
hook_IPO_INIT_EXP_OBS_D hook_IPO_INIT_EXP_OBS_W hook_IPO_INIT_EXP_RDX_D hook_IPO_INIT_EXP_RDX_W	This hook is for calling steps for initial batch exports.
hook_IPO_POST_BATCH_D hook_IPO_POST_BATCH_W	This hook is for calling steps after the batch has run.
hook_IPO_POST_DATA_IMP_OBS_D hook_IPO_POST_DATA_IMP_OBS_W hook_IPO_POST_DATA_IMP_RDX_D hook_IPO_POST_DATA_IMP_RDX_W	This hook is for the calling steps using any import of application-specific data interfaces after the calc steps.
hook_IPO_POST_EXP_OBS_D hook_IPO_POST_EXP_OBS_W hook_IPO_POST_EXP_RDX_D hook_IPO_POST_EXP_RDX_W	This hook is for the calling steps using any exports after the batch aggregations.
hook_IPO_PRE_BATCH_D hook_IPO_PRE_BATCH_W	This hook is for calling steps prior to the batch being run.
hook_IPO_PRE_DATA_IMP_OBS_D hook_IPO_PRE_DATA_IMP_OBS_W hook_IPO_PRE_DATA_IMP_RDX_D hook_IPO_PRE_DATA_IMP_RDX_W	This hook is for the calling steps using any import of application-specific data interfaces.
hook_IPO_PRE_EXP_OBS_D hook_IPO_PRE_EXP_OBS_W hook_IPO_PRE_EXP_RDX_D hook_IPO_PRE_EXP_RDX_W	This hook is for calling steps prior to exports.
hook_IPO_WB_BUILD_D hook_IPO_WB_BUILD_W	This hook is for the calling steps specific to workbook refresh or build.

Table 9-4 Boolean Scalar Measures

Boolean Scalar Measure	Description
appfalrton_CF_	This measure is set by the plug-in only. `_CF_` needs to be replaced by level number.
cslpeakalrton_CF_	This measure is set by the plug-in only. `_CF_` needs to be replaced by level number.
flysalrton_CF_	This measure is set by the plug-in only. `_CF_` needs to be replaced by level number.
fralrton_CF_	This measure is set by the plug-in only. `_CF_` needs to be replaced by level number.
runnewitembatch	This measure is defaulted to true. Set it to false if new item is not configured or user would like to skip new item batch for pre-forecast batch.
runfrcst_CF_	This measure is defaulted to true. Set it to false if customer would like to avoid running forecast on certain final level. `_CF_` needs to be replaced by level number.
runnewitem_CF_	This measure is defaulted to true. Set it to false if customer would like to avoid incorporate new item forecast on certain final level. `_CF_` needs to be replaced by level number.
runrulgeligga_CF_	This measure is defaulted to false. If enabled, this makes sure that the only forecastable items are handled by the business rule engine. `_CF_` needs to be replaced by level number.

Batch Control Samples

The following sections list samples of batch control processes.

batch_exec_list.txt

```
# unpack data file before data load
hook_pre_load | unpack      | rdf_hier.tar.gz
hook_pre_load | unpack      | rdf_meas.tar.gz


# load custom measures after GA hier and measure load
hook_pre_post_data_load | measload    | c_weeklyLoad


# calculate outlier indicator used in preprocess using custom rules
hook_ppsindicator | calc | c_outlier_calc


# calculate custom approval alerts after GA approval alerts
hook_frcst_alert07 | exec | c_calc_cust_alerts


# custom export
hook_post_export | measexport | c_export_promoeffects
c_calc_cust_alerts | calc |c_custalert1
c_calc_cust_alerts | calc |c_custalert2
```

batch_calc_list.txt

```
#outlier calculation
c_outlier_calc | G | GROUP | c_HBICalcTodayIdx
c_outlier_calc | G | GROUP | c_dataprocess
c_outlier_calc | G | GROUP | c_calc_outlier


#custom approval alerts calculation
c_custalert1 | G | GROUP | c_custalert1
c_custalert2 | G | GROUP | c_custalert2
```

batch_loadmeas_list.txt

```
# load custom measure
c_weeklyLoad | M | c_ActiveItem
c_weeklyLoad | M | c_DisContinue
```

batch_exportmeas_list.txt

```
# export custom measure
c_export_promoeffects|O|promoeffects.csv.dat
c_export_promoeffects|X|storsku_lprm
c_export_promoeffects|F|c_ExportMask
c_export_promoeffects|S|ftp
c_export_promoeffects|M|prmbldeff07
```

Below sections describe Batch Control details that are specific to MFP:

The following table describes the Custom Hooks available in the batch process if the customer is scheduling jobs directly through the OAT.

Table 9-5 Custom Hooks in the Batch Process to Directly Run from OAT

Hook	Description
hook_postbuild	This hook is added at the end of the `postbuild` batch, which runs after the initial domain build.
hook_postpatch	This hook is added at the end of the service patch process, which runs after the service patch.
hook_batch_daily_pre	This hook is added before the daily batch process.
hook_batch_daily_post	This hook is added at the end of the daily batch process before the dashboard build.
hook_batch_weekly_pre	This hook is added before the weekly batch process.
hook_batch_weekly_post	This hook is added at the end of the weekly batch process before the workbook refresh and segment build.

If the customer is using the JOS/POM flow schedule to schedule jobs in MFP, then the following hooks can be used. The MFP JOS/POM job flow is connected to use the same set names, like the hooks shown in the following table without `hook_*` in it and in turn calls each of the corresponding hooks. So the customer can easily customize their MFP batch flow based on their needs by simply changing the hooks or adding additional steps to the existing, pre-configured hooks.

The naming convention followed is:

- `_RDX` is used for any integration step using RDX.
- `_OBS` is used for any steps using Object Storage.
- `_D` is for jobs that run daily.
- `_W` is for jobs that run weekly.

Table 9-6 Custom Hooks in the Batch Process if JOS/POM is Used to Schedule the Flow

Hook	Description
hook_MFP_PRE_EXP_RDX_D	This hook is for the calling steps using the Daily Export Interfaces to RDX as soon as the batch starts.
hook_MFP_PRE_EXP_OBS_D	This hook is for the calling steps using the Daily Export Interfaces to Object Storage as soon as the batch starts.
hook_MFP_PRE_EXP_RDX_W	This hook is for calling steps using the Weekly Export Interfaces to RDX as soon as the batch starts.
hook_MFP_PRE_EXP_OBS_W	This hook is for the calling steps using the Weekly Export Interfaces to Object Storage as soon as the batch starts.
hook_MFP_COM_HIER_IM P_RDX_D	This hook is for the calling steps using any Daily Import of common hierarchies from RDX.
hook_MFP_COM_HIER_IM P_OBS_D	This hook is for the calling steps using any Daily Import of common hierarchies from Object Storage.

Table 9-6 (Cont.) Custom Hooks in the Batch Process if JOS/POM is Used to Schedule the Flow

Hook	Description
hook_MFP_COM_HIER_IM P_RDX_W	This hook is for the calling steps using any Weekly Import of common hierarchies from RDX.
hook_MFP_COM_HIER_IM P_OBS_W	This hook is for the calling steps using any Weekly Import of common hierarchies from Object Storage.
hook_MFP_COM_DATA_IM P_RDX_D	This hook is for the calling steps using any Daily Import of common data interfaces from RDX.
hook_MFP_COM_DATA_IM P_OBS_D	This hook is for the calling steps using any Daily Import of common data interfaces from Object Storage.
hook_MFP_COM_DATA_IM P_RDX_W	This hook is for the calling steps using any Weekly Import of common data interfaces from RDX.
hook_MFP_COM_DATA_IM P_OBS_W	This hook is for the calling steps using any Weekly Import of common data interfaces from Object Storage.
hook_MFP_HIER_IMP_RD X_D	This hook is for the calling steps using any Daily Import of application-specific hierarchies from RDX.
hook_MFP_HIER_IMP_OB S_D	This hook is for the calling steps using any Daily Import of application-specific hierarchies from Object Storage.
hook_MFP_HIER_IMP_RD X_W	This hook is for the calling steps using any Weekly Import of application-specific hierarchies from RDX.
hook_MFP_HIER_IMP_OB S_W	This hook is for the calling steps using any Weekly Import of application-specific hierarchies from Object Storage.
hook_MFP_PRE_DATA_IMP_RDX_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from RDX.
hook_MFP_PRE_DATA_IMP_OBS_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from Object Storage.
hook_MFP_PRE_DATA_IMP_RDX_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from RDX.
hook_MFP_PRE_DATA_IMP_OBS_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from Object Storage.
hook_MFP_BATCH_AGG_D	This hook is for the calling steps doing any regular daily batch aggregation after hierarchy and data loads.
hook_MFP_BATCH_AGG_ W	This hook is for the calling steps doing any regular weekly batch aggregation after hierarchy and data loads.
hook_MFP_POST_DATA_IM P_RDX_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from RDX after the calc steps.
hook_MFP_POST_DATA_IM P_OBS_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from Object Storage after the calc steps.
hook_MFP_POST_DATA_IM P_RDX_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from RDX after the calc steps.
hook_MFP_POST_DATA_IM P_OBS_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from Object Storage after the calc steps.
hook_MFP_POST_EXP_RD X_D	This hook is for the calling steps using any Daily Exports to RDX after the batch aggs.

Table 9-6 (Cont.) Custom Hooks in the Batch Process if JOS/POM is Used to Schedule the Flow

Hook	Description
hook_MFP_POST_EXP_OB S_D	This hook is for the calling steps using any Daily Exports to Object Storage after the batch aggs.
hook_MFP_POST_EXP_RD X_W	This hook is for the calling steps using any Weekly Exports to RDX after the batch aggs.
hook_MFP_POST_EXP_OB S_W	This hook is for the calling steps using any Weekly Exports to Object Storage after the batch aggs.
hook_MFP_WB_BUILD_D	This hook is for the calling steps specific to workbook refresh or build in the daily cycle.
hook_MFP_WB_BUILD_W	This hook is for the calling steps specific to workbook refresh or build in the weekly cycle.

Boolean Scalar Measures for Flow Control

The following table describes the Boolean Scalar measures.

Table 9-7 Boolean Scalar Measures

Boolean Scalar Measure	Description
drdvrmsb	This measure is defaulted to true. Set it to true if MFP is integrated with RMF CS.
drdvrdxb	This measure is defaulted to false. Set it to true enable RAP integration for hierarchy and transaction data.
drdvexpdb	This measure is defaulted to true. If set to false, it will skip exporting the standard exports in the daily batch.
drdvexpwb	This measure is defaulted to true. If set to false, it will skip exporting the standard exports in the weekly batch.

Batch Control Samples

The following sections show samples of the batch control processes.

batch_exec_list.txt

```
# Load a custom hierarchy, measure before weekly
batch hook_batch_weekly_pre |hierload |suph~0~N
hook_batch_weekly_pre |measload |c_load_vndr

# Run Batch calc and new custom exports after end of weekly batch
hook_batch_weekly_post |calc |c_calc_vndr
hook_batch_weekly_post |exportmeasure |c_exp_vndr
```

batch_calc_list.txt

```
# Run newly added custom calc rule group in
batch c_calc_vndr | G | GROUP | c_batch_agg_vndr
```

batch_loadmeas.txt

```
# Load custom measure
c_load_vndr | M |c_drtyvndrfndr
```

batch_exportmeas.txt

```
# Export custom measure
c_exp_vndr|O|vendo_plan.csv.dat
c_exp_vndr|X|storsclsweek
c_exp_vndr|F|c_exportmask
c_exp_vndr|M|c_mpcpvndrplan
```

Below sections describes Batch Control details that are specific to AP:

The following table describes the Custom Hooks available in the batch process.

Table 9-8 Custom Hooks in the Batch Process

Hook	Description
hook_postbuild_pre	This hook is added at the beginning of the postbuild batch which runs after the initial domain build.
hook_postbuild_post	This hook is added at the end of the postbuild batch which runs after the initial domain build.
hook_postpatch	This hook is added at the end of the service patch process which runs after the service patch.
hook_batch_daily_pre	This hook is added before the daily batch process.
hook_batch_daily_post	This hook is added at the end of daily batch process before the dashboard build.
hook_batch_weekly_pre	This hook is added before the weekly batch process.
hook_batch_weekly_post	This hook is added at the end of the weekly batch process before the workbook refresh and segment build.

If the customer is using the JOS/POM flow schedule to schedule jobs in AP, then the following hooks can be used. The AP JOS/POM job flow is connected to use the same set names similar to the hooks shown in the following table without `hook_*` in it and in turn calls each of the corresponding hooks. So the customer can easily customize their AP batch flow based on their needs by simply changing the hooks or adding additional steps to the existing pre-configured hooks.

The naming convention followed is:

- `_RDX` that is used for any integration step using RDX.
- `_OBS` is used for any steps using Object Storage.
- `_D` is for jobs that runs daily.
- `_W` is for jobs that runs weekly.

Table 9-9 Custom Hooks in the Batch Process

Hook	Description
hook_AP_PRE_EXP_RDX_D	This hook is for the calling steps using the Daily Export Interfaces to RDX as soon as the batch starts.
hook_AP_PRE_EXP_OBS_D	This hook is for the calling steps using the Daily Export Interfaces to Object Storage as soon as the batch starts.
hook_AP_PRE_EXP_RDX_W	This hook is for calling steps using the Weekly Export Interfaces to RDX as soon as the batch starts.
hook_AP_PRE_EXP_OBS_W	This hook is for the calling steps using the Weekly Export Interfaces to Object Storage as soon as the batch starts.
hook_AP_COM_HIER_IMP_RDX_D	This hook is for the calling steps using any Daily Import of common hierarchies from RDX.
hook_AP_COM_HIER_IMP_OBS_D	This hook is for the calling steps using any Daily Import of common hierarchies from Object Storage.
hook_AP_COM_HIER_IMP_RDX_W	This hook is for the calling steps using any Weekly Import of common hierarchies from RDX.
hook_AP_COM_HIER_IMP_OBS_W	This hook is for the calling steps using any Weekly Import of common hierarchies from Object Storage.
hook_AP_COM_DATA_IMP_RDX_D	This hook is for the calling steps using any Daily Import of common data interfaces from RDX.
hook_AP_COM_DATA_IMP_OBS_D	This hook is for the calling steps using any Daily Import of common data interfaces from Object Storage.
hook_AP_COM_DATA_IMP_RDX_W	This hook is for the calling steps using any Weekly Import of common data interfaces from RDX.
hook_AP_COM_DATA_IMP_OBS_W	This hook is for the calling steps using any Weekly Import of common data interfaces from Object Storage.
hook_AP_HIER_IMP_RDX_D	This hook is for the calling steps using any Daily Import of application-specific hierarchies from RDX.
hook_AP_HIER_IMP_OBS_D	This hook is for the calling steps using any Daily Import of application-specific hierarchies from Object Storage.
hook_AP_HIER_IMP_RDX_W	This hook is for the calling steps using any Weekly Import of application-specific hierarchies from RDX.
hook_AP_HIER_IMP_OBS_W	This hook is for the calling steps using any Weekly Import of application-specific hierarchies from Object Storage.
hook_AP_PRE_DATA_IMP_RDX_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from RDX.
hook_AP_PRE_DATA_IMP_OBS_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from Object Storage.
hook_AP_PRE_DATA_IMP_RDX_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from RDX.
hook_AP_PRE_DATA_IMP_OBS_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from Object Storage.
hook_AP_BATCH_AGG_D	This hook is for the calling steps doing any regular daily batch aggregation after hierarchy and data loads.
hook_AP_BATCH_AGG_W	This hook is for the calling steps doing any regular weekly batch aggregation after hierarchy and data loads.
hook_AP_POST_DATA_IMP_RDX_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from RDX after the calc steps.

Table 9-9 (Cont.) Custom Hooks in the Batch Process

Hook	Description
hook_AP_POST_DATA_IMP_OBS_D	This hook is for the calling steps using any Daily Import of application-specific data interfaces from Object Storage after the calc steps.
hook_AP_POST_DATA_IMP_RDX_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from RDX after the calc steps.
hook_AP_POST_DATA_IMP_OBS_W	This hook is for the calling steps using any Weekly Import of application-specific data interfaces from Object Storage after the calc steps.
hook_AP_POST_EXP_RDX_D	This hook is for the calling steps using any Daily Exports to RDX after the batch aggs.
hook_AP_POST_EXP_OBS_D	This hook is for the calling steps using any Daily Exports to Object Storage after the batch aggs.
hook_AP_POST_EXP_RDX_W	This hook is for the calling steps using any Weekly Exports to RDX after the batch aggs.
hook_AP_POST_EXP_OBS_W	This hook is for the calling steps using any Weekly Exports to Object Storage after the batch aggs.
hook_AP_WB_BUILD_D	This hook is for the calling steps specific to workbook refresh or build in the daily cycle.
hook_AP_WB_BUILD_W	This hook is for the calling steps specific to workbook refresh or build in the weekly cycle.

Boolean Scalar Measures for Flow Control

The following table describes the Boolean Scalar measures.

Table 9-10 Boolean Scalar Measures

Boolean Scalar Measure	Description
drdvrmsb	This measure is defaulted to true. Set it to true if AP is integrated with RMF CS.
drdvexpdb	This measure is defaulted to true. If set to false, it will skip exporting the standard exports in the daily batch.
drdvexpwb	This measure is defaulted to true. If set to false, it will skip exporting the standard exports in the weekly batch.

Batch Control Samples

The following sections show samples of the batch control processes.

batch_exec_list.txt

```
# Load a custom hierarchy, measure before weekly batch
hook_batch_weekly_pre |hierload |suph~0~N
hook_batch_weekly_pre |measload |c_load_vndr

# Run Batch calc and new custom exports after end of weekly batch
```

```
hook_batch_weekly_post |calc |c_calc_vndr
hook_batch_weekly_post |exportmeasure |c_exp_vndr
```

batch_calc_list.txt

```
# Run newly added custom calc rule group in batch
c_calc_vndr | G | GROUP | c_batch_agg_vndr
```

batch_loadmeas.txt

```
# Load custom measure
c_load_vndr | M |c_drtyvndrfndr
```

batch_exportmeas.txt

```
# Export custom measure
c_exp_vndr|O|vendo_plan.csv.dat
c_exp_vndr|X|storsclsweek
c_exp_vndr|F|c_expmask
c_exp_vndr|M|c_mpcpvndrpln
```

Programmatic Extensibility of RPASCE Through Innovation Workbench

Innovation Workbench is the first choice for programmatic extensibility of RAP applications.

This section describes how Innovation Workbench (IW) can be used to extend RPASCE.

IW provides the ability for customers to upload and use custom PL/SQL functions and procedures within the RPASCE framework. These customer-supplied PL/SQL functions and procedures will, by default, have read access to all the metadata tables and fact data that is present across different RPASCE applications within PDS, such as MFP, IPOCS-Demand Forecasting, AP, and so on.

Facts that are marked as customer-managed in the application configuration will additionally have write access and can be modified by the customer-supplied code. The ability to change fact data is a deliberate opt-in, as any data modification made by the custom PL/SQL must conform to the RPASCE norms to be successful.

RPASCE provides a helper PL/SQL package: `rp_g_rpas_helper_pkg`. This is the package that the custom PL/SQL functions and procedures should use. These will help simplify commonly used tasks, such as looking up the fact table name for a fact or finding the NA value of a FACT.

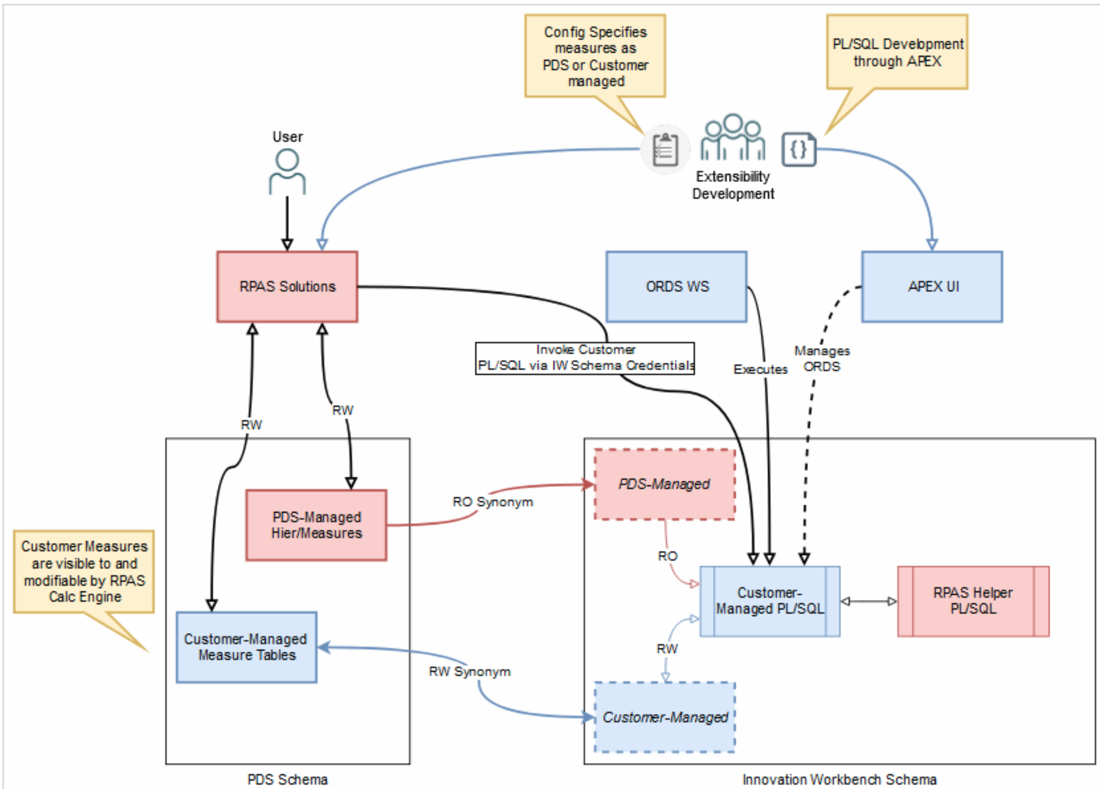
Finally, the applications will be able to invoke the custom PL/SQL functions and procedures from within the RPASCE rules and expressions framework using the new special expression `execplsql`.

Architectural Overview

The figures in this section describe how the IW schema fits into the PDS and RAP contexts respectively.

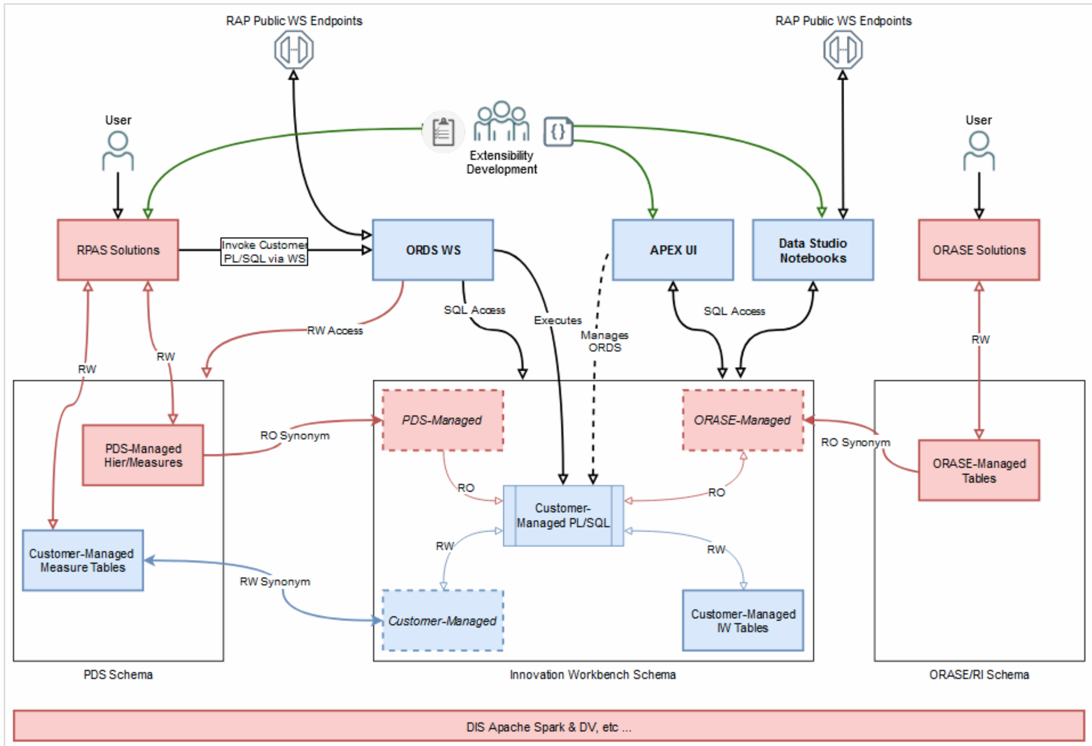
Innovation Workbench from an RPASCE Context

The figure below shows a high-level architecture diagram of the IW schema in an RPASCE context. The interaction between the PDS schema and IW schema and how users interact with the IW schema is captured in this figure.



Innovation Workbench from a RAP Context

The figure below depicts how the IW schema fits into the larger RAP context, including AIF. Within the IW schema, the customer has access to data from both AIF and PDS configurations, which allows for the development of innovative extensions. These can be invoked from RPASCE, APEX, Data Studio notebooks, or an external context through ORDS web services.



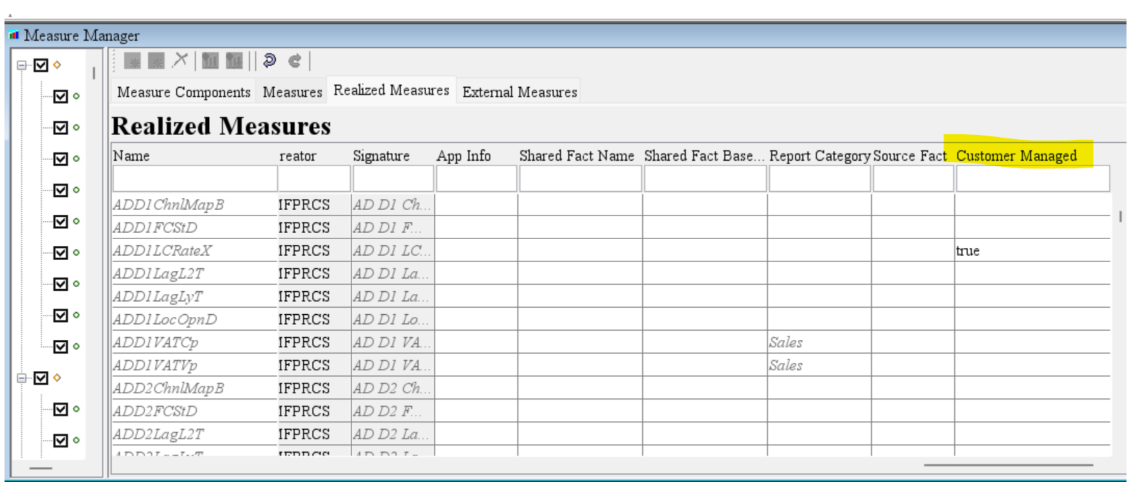
RPASCE Configuration Tools Changes

Measure Properties

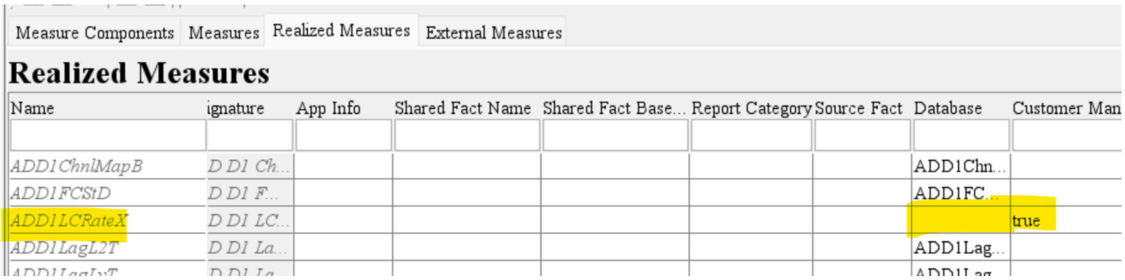
If the customer-provided PL/SQL functions and procedures require write-access to any RPASCE measures, then they must be marked as "Customer-Managed" in the application configuration.

In ConfigTools Workbench, a new column **Customer Managed** is added to the Measure Definition Table. This new column is defaulted to empty, which means **false**.

To mark a measure as customer managed the value should be changed to **true**.



The "customer-managed" measures must have a database field specified, otherwise an error will be thrown.



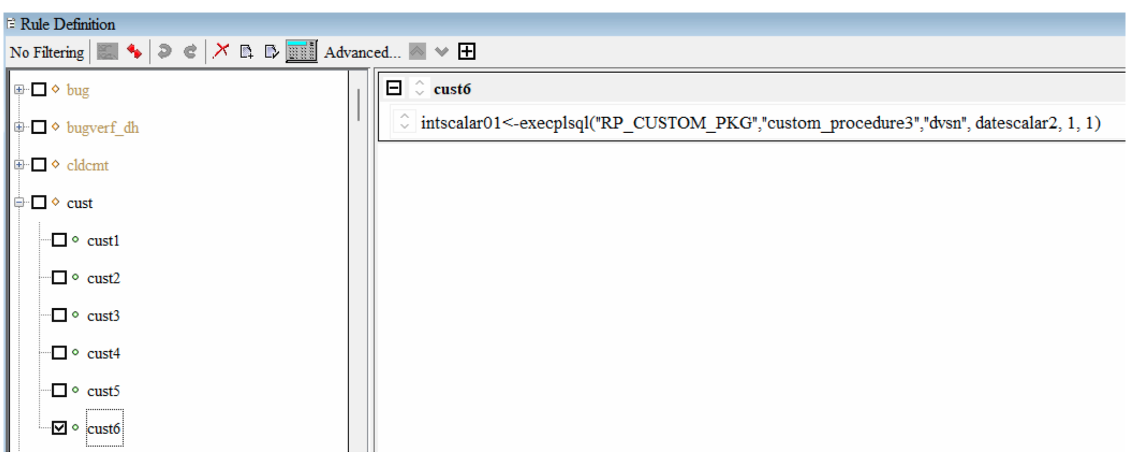
> **Note:**
>
> The "customer-managed" measures cannot be used in cycle groups and the left-hand side of special expressions because these measures need to be in the same fact group. Making part of these measures as customer-managed measures/facts will split this fact group because customer-managed measures are assigned to a separate fact group.

Rules and Expressions

To invoke the uploaded PL/SQL functions and procedures, add rules with the special expression `execplsql` into the RPASCE rule definitions in the configtools. For more details about the special expression `execplsql` please refer to the section **Special Expression - execplsql** in this document.

Example

In the example below, a rule containing `execplsql` is added to rule group `cust6`.



One requirement is that a `cmf` rulegroup must have only `plsqlexec` rules. It is not possible to mix other kinds of rules with the `plsqlexec` rule. There can be many `plsqlexec` rules in the same rulegroup. Also please make sure keep only one expression in each rule.

Integration Configuration

The Integration Configuration Tool will have a new column **Customer-Managed** for the Integration Map table. The integration configuration is generated internally and is only shown here for information purposes.

```
<integration_map>
  <entry>
    <fact>ADDVChWhMapT</fact>
    <domain>mfpcs</domain>
    <measure>ADDVChWhMapT</measure>
    <outbound>N</outbound>
    <customer-managed>Y</customer-managed>
  </entry>
</integration_map>
```

RPASCE Special Expression - execplsql

The special expression `execplsql` provides the ability to invoke the customer uploaded PL/SQL functions and procedures from within the RPASCE rules and expressions framework. Both functions and procedures are supported. Also `execplsql` is variadic and can take an arbitrary number of arguments that can be of different types according to the PL/SQL signature. The number and type depend on the signature of the function or procedures being executed. The first two arguments to `execplsql` are reserved to indicate the package name and the function or procedure name to be executed.

Example

```
drdvsrcti<-execplsql("RP_CUSTOM_PKG","sum",drdvsrctt, adhdlcratet,
add2locopnd)
```

In this example the LHS measure `drdvsrcti` is a scalar integer measure. It will be set to the integer value returned by the function named `sum` in the customer uploaded package `RP_CUSTOM_PKG`.

Arguments

LHS

The LHS measure must be a scalar integer measure. It will be set to the integer value returned by the customer-uploaded PL/SQL function or procedure. The integer value is meant to be a return code indicating the result of the procedure or function execution. In case of exceptions, RPASCE will set the LHS measure to a value of `-1` to indicate an error. If there are any exceptions or failures, then the logs will provide further information regarding the reason for the failure.

RHS

- First argument:

  The type of the first argument is `string`. It can either be a string constant or a scalar string measure. The first argument is the name of the customer-uploaded package. For more details regarding uploading custom packages refer to section Uploading Custom PL/SQL Packages.

- Second Argument:

  The type of the first argument is `string`. It can either be a string constant or a scalar string measure. The second argument is the name of a function or procedure within in the custom package specified as the first argument of `execplsql`. This function or procedure will be executed by the `execplsql` special expression when it is evaluated.

  If a function is being specified, make sure the return type is declared as a number in the PL/SQL function declaration.

  If a procedure is being specified, make sure there is exactly 1 out type parameter of type number in the PL/SQL procedure declaration.

Examples

Consider the PL/SQL function SUM present in the package `RP_CUSTOM_PKG`. To execute the `SUM` function in the RPASCE application batch, first upload `RP_CUSTOM_PKG` as described in the section Uploading Custom PL/SQL Packages. The PL/SQL function `SUM` is declared as below in the package `RP_CUSTOM_PKG`.

```
function SUM (lhsMeas IN VARCHAR2, rhsMeas1 IN VARCHAR2, rhsMeas2 IN
VARCHAR2) return number;
```

Here is a sample definition of the `SUM` function that adds 2 measures and writes the result to a third measure. Note that the measure `lhsMeas` is an `IN` type argument although the function `SUM` updates it. The measure `lhsMeas` must be marked as a customer managed measure as described in the Measure Properties subsection of the RPASCE Configuration Tools Changes section.

```
    FUNCTION sum (
        lhsmeas   IN VARCHAR2,
        rhsmeas1  IN VARCHAR2,
        rhsmeas2  IN VARCHAR2
    ) RETURN NUMBER IS
-- EXPR 1: lhsMeas = rhsMeas2 + rhsMeas1

        na_ut_lhsmeas  BINARY_DOUBLE := cell_dbl(lhsmeas, NULL);
        na_ut_rhsmeas1 BINARY_DOUBLE := cell_dbl(rhsmeas1, NULL);
        na_ut_rhsmeas2 BINARY_DOUBLE := cell_dbl(rhsmeas2, NULL);
        lhsfactgroup   VARCHAR2(4000);
        rhs1factgroup  VARCHAR2(4000);
        rhs2factgroup  VARCHAR2(4000);
        lhsfacttable   VARCHAR2(4000);
        rhs1facttable  VARCHAR2(4000);
        rhs2facttable  VARCHAR2(4000);
        stmt           VARCHAR2(8000);
    BEGIN
        rp_g_common_pkg.clear_facts(varchar2_table(lhsmeas));
        SELECT
            fact_group
        INTO lhsfactgroup
        FROM
            rp_g_fact_info_md
```

```
        WHERE
            fact_name = lhsmeas;

        SELECT
            fact_group
        INTO rhs1factgroup
        FROM
            rp_g_fact_info_md
        WHERE
            fact_name = rhsmeas1;

        SELECT
            fact_group
        INTO rhs2factgroup
        FROM
            rp_g_fact_info_md
        WHERE
            fact_name = rhsmeas2;

        lhsfacttable := 'rp_g_'
                        || lhsfactgroup
                        || '_ft';
        rhs1facttable := 'rp_g_'
                         || rhs1factgroup
                         || '_ft';
        rhs2facttable := 'rp_g_'
                         || rhs2factgroup
                         || '_ft';
        na_ut_lhsmeas := ( na_ut_rhsmeas2 + na_ut_rhsmeas1 );

        stmt := 'MERGE INTO '
                || lhsfacttable
                || ' lhs
    USING (
              SELECT
                  ( coalesce(rhsft01.partition_id,
rhsft02.partition_id) )               AS partition_id,
                  ( coalesce(rhsft01.dept_id,
rhsft02.dept_id) )                           AS dept_id,
                  ( coalesce(rhsft01.stor_id,
rhsft02.stor_id) )                           AS stor_id,
                  ( ( coalesce(rhsft02.'
                || rhsmeas2
                || ', '
                || na_ut_rhsmeas2
                || ') + coalesce(rhsft01.'
                || rhsmeas1
                || ', '
                || na_ut_rhsmeas1
                || ') ) ) AS '
                || lhsmeas
                || '
              FROM
                  (
                      SELECT
                          partition_id,
```

```
                      dept_id,
                      stor_id,
                    '
            || rhsmeas1
            || '
                FROM
                   '
            || rhs1facttable
            || '
              ) rhsft01
              FULL OUTER JOIN (
                  SELECT
                      partition_id,
                      dept_id,
                      stor_id,
                      '
            || rhsmeas2
            || '
                  FROM
                      '
            || rhs2facttable
            || '
              ) rhsft02 ON rhsft01.partition_id = rhsft02.partition_id
                           AND rhsft01.dept_id = rhsft02.dept_id
                           AND rhsft01.stor_id = rhsft02.stor_id
      )
rhs_final ON ( lhs.partition_id = rhs_final.partition_id
               AND lhs.dept_id = rhs_final.dept_id
               AND lhs.stor_id = rhs_final.stor_id )
WHEN MATCHED THEN UPDATE
SET lhs.'
            || lhsmeas
            || '= nullif(rhs_final.'
            || lhsmeas
            || ', '
            || na_ut_lhsmeas
            || ') DELETE
WHERE
    rhs_final.'
            || lhsmeas
            || ' =  '
            || na_ut_lhsmeas
            || '
WHEN NOT MATCHED THEN
INSERT (
    lhs.partition_id,
    lhs.dept_id,
    lhs.stor_id,
    lhs.'
            || lhsmeas
            || ' )
VALUES
    ( rhs_final.partition_id,
      rhs_final.dept_id,
      rhs_final.stor_id,
    nullif(rhs_final.'
```

```
                    || lhsmeas
                    || ', '
                    || na_ut_lhsmeas
                    || ') )
        WHERE
            rhs_final.'
                    || lhsmeas
                    || ' != '
                    || na_ut_lhsmeas;

            dbms_output.put_line(stmt);
            EXECUTE IMMEDIATE stmt;
            COMMIT;
            RETURN 0;
        END sum;
```

Now to execute this `SUM` function from the application batch, add the rule below to application configuration as described in the Rules and Expressions subsection of RPASCE Configuration Tools Changes section. Add the rule group containing the rule to the batch control files as described in the section RPASCE Batch Control File Changes. Patch the application with the updated configuration and batch control files.

```
drdvsrcti<-execplsql("RP_CUSTOM_PKG","sum",drdvsrctt, adhdlcratet,
add2locopnd)
```

Here all 3 measures are placeholder scalar string measures that will point to the actual real measures that are being summed.

In this example, the input scalar measures are mapped as follows:

- **drdvsrctt:** lpwpsellthrmn - dept_stor - customer managed (LHS measure)

  Label: Wp Sell Thru R % Min Threshold
- **adhdlcratet:** lpwprtnmn - dept_stor (RHS1)

  Label: Wp Returns R % Min Threshold
- **add2locopnd:** lpwprtnmx - dept_stor (RHS2)

  Label: Wp Returns R % Max Threshold

Alternately, the rule could have been configured as below. However, that would mean that it is not possible to change the input measures as part of the batch. It will need a patch to update the input measures to the `SUM` procedure.

> **Note:**
>
> The measures are in quotes as they are passed to PL/SQL as string constants. If the quotes are missing, then RPASCE will throw an error indicating that it is not possible to invoke `execplsql` using non-scalar measures.

```
drdvsrcti<-execplsql("RP_CUSTOM_PKG","sum",'lpwpsellthrmn' ,
'lpwprtnmn' ,'lpwprtnmx' )
```

Execute this rule group through batch and build a measure analysis workbook with the involved measures. It can then be verified that the `SUM` evaluated correctly.

Location / Measure	1011 Ottawa	1012 Toronto	1013 Montreal
Wp Returns R % Max Threshold	5.00	5.00	2.00
Wp Returns R % Min Threshold	5.00	5.00	3.00
Wp Sell Thru R % Min Threshold	10.00	10.00	5.00

If measure `lhsMeas` is not specified as a customer-managed measure in the application configuration, then the error below will be thrown when `execplsql` is evaluated.

```
<E OCI_ERROR - (1031):
<E SQL Sid 'rpas_iw_conn' ORA-01031: insufficient privileges
```

The following examples demonstrate `execplsql` and how a special expression can invoke PL/SQL with a variable number and type of input arguments.

```
 intscalar01<-execplsql("RP_CUSTOM_PKG","custom_procedure1","dvsn", true, 1, 1)
```

```
 intscalar01<-execplsql("RP_CUSTOM_PKG","custom_procedure2","dvsn", 1123.5813, 23, -1)
```

```
 intscalar01<-execplsql("RP_CUSTOM_PKG","custom_function1","dvsn", true, 1)
```

```
 intscalar01<-execplsql("RP_CUSTOM_PKG","custom_function2","dvsn", 1)
```

```
 intscalar01<-execplsql("RP_CUSTOM_PKG","custom_function2",strscalar1, intscalar02)
```

```
 intscalar01<-execplsql("RP_CUSTOM_PKG","custom_procedure3","dvsn", datescalar2, 1, 1)
```

The PL/SQL counterparts are defined, through very simple demonstration code, in the example custom package below.

rp_custom_pdg.pkb

```
create or replace package body RP_CUSTOM_PKG is

procedure custom_procedure1(arg1 in varchar2, arg2 in CHAR, arg3 in number,
ret out number )
is
begin
   ret:=23;
end custom_procedure1;

procedure custom_procedure2(arg1 in varchar2, arg2 in BINARY_DOUBLE, arg3 in
number, ret out number )
is
begin
   ret:=23;
end custom_procedure2;

procedure custom_procedure3(arg1 in varchar2, arg2 in BINARY_DOUBLE, arg3 in
timestamp, ret out number )
is
begin
  dbms_output.put_line('arg3: ' || arg3);
   ret:=23;
end custom_procedure3;

function custom_function1(arg1 in varchar2, arg2 in CHAR, arg3 in number)
return number
is
 ret integer;
begin
   ret:=23;
   return ret;
end custom_function1;

function custom_function2(arg1 in varchar2, arg3 in number) return number
is
  ret number;
begin
   ret:=23;
   return ret;
end custom_function2;


function SUM
(lhsMeas IN VARCHAR2,
rhsMeas1 IN VARCHAR2,
rhsMeas2 IN VARCHAR2) return number
is
-- EXPR 1: lhsMeas = rhsMeas2 + rhsMeas1

    na_ut_lhsMeas  BINARY_DOUBLE := cell_dbl(lhsMeas, NULL);
    na_ut_rhsMeas1 BINARY_DOUBLE := cell_dbl(rhsMeas1, NULL);
    na_ut_rhsMeas2 BINARY_DOUBLE := cell_dbl(rhsMeas2, NULL);
    lhsFactGroup varchar2(4000);
```

```
    rhs1FactGroup varchar2(4000);
    rhs2FactGroup varchar2(4000);
    lhsFactTable varchar2(4000);
    rhs1FactTable varchar2(4000);
    rhs2FactTable varchar2(4000);
    stmt varchar2(8000);
BEGIN

    rp_g_common_pkg.clear_facts(varchar2_table(lhsMeas));
    select fact_group into lhsFactGroup from RP_G_FACT_INFO_MD where
FACT_NAME = lhsMeas;
    select fact_group into rhs1FactGroup from RP_G_FACT_INFO_MD where
FACT_NAME = rhsMeas1;
    select fact_group into rhs2FactGroup from RP_G_FACT_INFO_MD where
FACT_NAME = rhsMeas2;
    lhsFactTable := 'rp_g_' || lhsFactGroup || '_ft';
    rhs1FactTable := 'rp_g_' || rhs1FactGroup || '_ft';
    rhs2FactTable := 'rp_g_' || rhs2FactGroup || '_ft';
    na_ut_lhsMeas := ( na_ut_rhsMeas2 + na_ut_rhsMeas1 );

    -- UPDATE rp_g_fact_info_md
    -- SET
    --     table_na =
    --         CASE lower(fact_name)
    --             WHEN 'b' THEN
    --                 to_char(na_ut_lhsMeas)
    --         END
    -- WHERE
    --     lower(fact_name) IN ( lhsMeas );

    stmt :=      'MERGE INTO ' || lhsFactTable || ' lhs
    USING (
              SELECT
                  ( coalesce(rhsft01.partition_id,
rhsft02.partition_id) )               AS partition_id,
                  ( coalesce(rhsft01.dept_id,
rhsft02.dept_id) )                           AS dept_id,
                  ( coalesce(rhsft01.stor_id,
rhsft02.stor_id) )                           AS stor_id,
                  ( ( coalesce(rhsft02.' || rhsMeas2 || ', ' ||
na_ut_rhsMeas2 || ') + coalesce(rhsft01.' || rhsMeas1 || ', ' ||
na_ut_rhsMeas1 || ') ) ) AS '|| lhsMeas||'
              FROM
                  (
                      SELECT
                          partition_id,
                          dept_id,
                          stor_id,
                        '||  rhsMeas1 ||'
                    FROM
                        ' || rhs1FactTable || '
                  ) rhsft01
                  FULL OUTER JOIN (
                      SELECT
                          partition_id,
                          dept_id,
```

```
                            stor_id,
                            ' || rhsMeas2 || '
                        FROM
                            ' || rhs2FactTable || '
                    ) rhsft02 ON rhsft01.partition_id = rhsft02.partition_id
                                 AND rhsft01.dept_id = rhsft02.dept_id
                                 AND rhsft01.stor_id = rhsft02.stor_id
            )
    rhs_final ON ( lhs.partition_id = rhs_final.partition_id
                   AND lhs.dept_id = rhs_final.dept_id
                   AND lhs.stor_id = rhs_final.stor_id )
    WHEN MATCHED THEN UPDATE
    SET lhs.' || lhsMeas || '= nullif(rhs_final.' || lhsMeas || ', ' ||
na_ut_lhsMeas || ') DELETE
    WHERE
        rhs_final.' || lhsMeas || ' =  ' || na_ut_lhsMeas || '
    WHEN NOT MATCHED THEN
    INSERT (
        lhs.partition_id,
        lhs.dept_id,
        lhs.stor_id,
        lhs.' || lhsMeas || ' )
    VALUES
        ( rhs_final.partition_id,
          rhs_final.dept_id,
          rhs_final.stor_id,
        nullif(rhs_final.' || lhsMeas || ', ' || na_ut_lhsMeas || ') )
    WHERE
        rhs_final.' || lhsMeas || ' != ' || na_ut_lhsMeas ;


 DBMS_OUTPUT.PUT_LINE (stmt);

execute immediate stmt;

commit;

return 0;

END SUM;



end RP_CUSTOM_PKG;
```

rp_custom_pkg.pks

```
create or replace package RP_CUSTOM_PKG is

procedure custom_procedure1(arg1 in varchar2, arg2 in CHAR, arg3 in number,
ret out number );
procedure custom_procedure2(arg1 in varchar2, arg2 in BINARY_DOUBLE, arg3 in
number, ret out number );
procedure custom_procedure3(arg1 in varchar2, arg2 in BINARY_DOUBLE, arg3 in
timestamp, ret out number );
```

```
function custom_function1(arg1 in varchar2, arg2 in CHAR, arg3 in number)
return number ;
function custom_function2(arg1 in varchar2, arg3 in number) return number;
function SUM (lhsMeas IN VARCHAR2, rhsMeas1 IN VARCHAR2, rhsMeas2 IN
VARCHAR2) return number;
end RP_CUSTOM_PKG;
```

Limitations

Boolean arguments must be recast as character types, the PL/SQL function or procedure should declare them as a `CHAR` type. RPASCE will set the char to `T` for true and `F` for false. On the expression side it is handled similarly to how boolean types are handled by an RPASCE expression. Pass a scalar boolean measure or boolean constant (true or false) to the `execplsql` special expression.

Validations and Common Error Messages

Common validation error messages are documented in the table below. However, there can be other kinds of errors (for example, unexpected privilege related errors).

Error message	Reason
Output argument type of the procedure must be NUMBER.	The procedure specified has an `OUT` parameter, but it is not of the `NUMBER` type.
Function or procedure not found.	Check the name of the function or procedure in the custom package. Could be case mismatch or privilege issue too.
No output type found.	There is no `OUT` parameter associated with the procedure specified.
Only 1 output argument allowed.	There is more than one `OUT` parameter associated with the procedure specified.
Input arg not a scalar measure.	Only scalar string measures and string constants are allowed as input. Check whether quotes are missing in case of constants.
Expecting <type> but received <type>.	Mismatch in argument type between PL/SQL function or procedure signature and the arguments passed to `execplsql` expression.
Only 1 measure allowed on the LHS.	There can be only 1 measure on the LHS of the `execplsql` expression and it must be a scalar `int` measure.
RHS size must be more than 2. First 2 args are package name and procedure name.	Not enough arguments passed in to the `execplsql` expression. There must be at least 2: package name and procedure/function name.
LHS must be a measure	Found LHS to be a constant instead of a measure.
LHS must be a scalar measure.	There is an LHS measure but it is not scalar.
LHS measure must be of type integer.	There is an LHS measure and it is not scalar, but it is not of type `integer`.
Number of input args <number> does not match the procedure signature number <number>.	Mismatch in number of arguments between PL/SQL function or procedure signature and the arguments passed to `execplsql` expression.

Error message	Reason
ExecPLSQLExpression incrementalEval not supported!	If other errors are bypassed and `execplsql` is used along with other the workbook calculation rules.
PL/SQL type <type> is not supported.	Unexpected type in the PL/SQL function or procedure signature.

RPASCE Batch Control File Changes

Invoking the CMF rulegroup from batch is done by adding the rulegroup to the batch control files. The example below shows how to add a CMF rule group to batch control files. There are no additional steps required here. Please refer to any GA application implementation guide for more information on adding rulegroups to batch control files.

File: batch_calc_list.txt

```
iw_sum | group | cust7
```

In this example, the calc set name is `iw_sum`, which is of type `group`, meaning it is executing a rule group. The third item is the rule group name, which is `cust7`. Rule group `cust7` has a CMF property set and contains `execplsql` rules.

File: batch_oat_list.txt

```
calc | iw_sum | IW Sum
```

Here the Batch Control Group Name is `calc` and the batch set name is `iw_sum`, meaning it will look at file `batch_calc_list.txt` for an entry named `iw_sum`. We already added it in the step above. The third item is the label, which shows up on the UI in the drop down when user tries to execute the batch calc group OAT task.

File: batch_exec_list.txt

```
iw_all | calc | iw_sum
```

Here `iw_all` is the Batch Set Name. Batch task type is `calc` and parameter is `iw_sum`. When `iw_all` is invoked, it will look for an entry named `iw_sum` in the `batch_calc_list.txt`. Please check the first step above for the entry in `batch_calc_list.txt`.

The `iw_all` can be made part of a daily batch as shown in the example below.

File: batch_exec_list.txt

```
# Daily Batch Cyle
batch_daily | exec   | *hook_batch_daily_pre
batch_daily | exec   | DRDVEXPDB ? export_all
batch_daily | exec   | batch_oo
batch_daily | exec   | *hook_batch_daily_post
batch_daily | exec   | re_daily
batch_daily | exec   | iw_all
```

RPASCE Deployment

The customer-managed PL/SQL functions and procedures are uploaded to the IW schema. For more information on uploading the custom packages, refer to the section Uploading Custom PL/SQL Packages.

During evaluation of the `execplsql` special expression, RPASCE switches to the IW schema user, to limit the scope of writable data access, and then executes the function or procedure. However, during application deploy and patch, RPASCE code grants the necessary privileges to the IW schema user. These grants ensure that the IW schema user can read all the fact tables and metadata tables in the PDS through synonyms, and write access is only provided to fact tables for the measures marked as customer-managed in the application configuration.

If the configuration is modified such that additional measures are now marked as customer managed, or if existing customer-managed measures are made non-customer-managed, then the application patch operation will update the privileges accordingly.

Uploading Custom PL/SQL Packages

Refer to Chapter 20 of *Oracle Retail AI Foundation Cloud Services Implementation Guide* for further details on interacting with IW schema (`RTLWSP01`).

RPASCE Helper Functions and API for IW

RPASCE provides a package `RP_G_RPAS_HELPER_PKG` with useful methods that can be called from custom code.

Following are the types that are available in `RP_G_RPAS_HELPER_PKG`:

```
TYPE t_pair IS RECORD(
            l_level varchar2(10),
            l_dim varchar2(10)
        );


TYPE dim_level_array is varray(8) of t_pair;


TYPE level_array is varray(50) of varchar2(10);
```

Following are the functions that are available in `RP_G_RPAS_HELPER_PKG`.

function get_fact_name(meas_name_in IN varchar2) return varchar2;

This function returns the fact name based on the measure name passed. Measure name is defined in Configuration.

```
declare
   l_fact varchar2(30);
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
end;
```

function get_na_value(fact_name_in IN varchar2) return varchar2;

Function returns NA Value for a fact.

```
declare
   l_fact     varchar2(30);
   l_na_value number;
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
   l_na_value := rp_g_rpas_helper_pkg.get_na_value(l_fact);
end;
```

function get_logical_space(fact_name_in IN varchar2) return number;

Function returns logical space for a fact. Logical space is the unpopulated space for a fact.

```
declare
   l_fact      varchar2(30);
   l_log_space number;
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
   l_log_space := rp_g_rpas_helper_pkg.get_logical_space(l_fact);
end;
```

function get_fact_group_name(fact_name_in IN varchar2) return varchar2;

Function returns the Fact Group name for a fact.

```
declare
   l_fact      varchar2(30);
   l_group     varchar2(30);
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
   l_group := rp_g_rpas_helper_pkg.get_fact_group_name(l_fact);
end;
```

function get_table_name(fact_name_in IN varchar2) return varchar2;

Function returns the Oracle table name that contains the fact as a column.

```
declare
   l_fact      varchar2(30);
   l_table     varchar2(30);
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
   l_table := rp_g_rpas_helper_pkg.get_table_name(l_fact);
end;
```

function get_number_of_partitions return integer;

Function returns the number of partitions in PDS.

```
declare
   l_parts     number;
```

```
begin
   l_parts := rp_g_rpas_helper_pkg.get_number_of_partitions;
end;
```

function get_partition_level return varchar2;

Function returns the level at which is PDS is partitioned (for example, `dept`,`clss`).

```
declare
   l_part_level     varchar2(30);
begin
   l_part_level := rp_g_rpas_helper_pkg.get_partition_level;
end;
```

function clear_fact(fact_name_in varchar2) return boolean;

Function will clear the data for the fact.

```
declare
   l_result     boolean;
begin
   l_result := rp_g_rpas_helper_pkg.clear_fact('drtynslsu');
end;
```

function get_base_intx(fact_name_in IN varchar2) return varchar2;

Function returns the base intersection of the fact.

```
declare
   l_fact      varchar2(30);
   l_intx      varchar2(30);
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
   l_intx := rp_g_rpas_helper_pkg.get_base_intx(l_fact);
end;
```

function intx_to_level(intx_in IN varchar2) return dim_level_array;

Function returns the levels and dimensions that are part of an intersection in an array.

```
declare
   l_fact      varchar2(30);
   l_intx      varchar2(30);
   l_array      dim_level_array;
begin
   l_fact := rp_g_rpas_helper_pkg.get_fact_name('drtynslsu');
   l_intx := rp_g_rpas_helper_pkg.get_base_intx(l_fact);
   l_array := rp_g_rpas_helper_pkg.intx_to_level(l_intx);
end;
```

function get_parent_levels(level_in IN varchar2) return level_array;

This returns all the parent levels of the passed level.

```
declare
   l_array      level_array;
begin
   l_array := rp_g_rpas_helper_pkg.get_parent_levels('styl');
end;
```

function get_child_levels(level_in IN varchar2) return level_array;

This returns all the child levels of the passed level.

```
declare
   l_array      level_array;
begin
   l_array := rp_g_rpas_helper_pkg.get_child_levels('styl');
end;
```

function is_higher_level(level1_in IN varchar2, level2_in IN varchar2) return boolean;

Function return true if `level1_in` is higher than `level2_in`. Otherwise false.

```
declare
    l_val      boolean;
begin
    l_val := rp_g_rpas_helper_pkg.is_higher_level('styl', 'dept');
end;
```

function is_lower_level(level1_in IN varchar2, level2_in IN varchar2) return boolean;

Function return true if `level1_in` is lower than `level2_in`. Otherwise false.

```
declare
    l_val      boolean;
begin
    l_val := rp_g_rpas_helper_pkg.is_lower_level('dept', 'styl');
end;
```

PL/SQL Best Practices

Number	PL/SQL Best Practice	Comments
1	Use the functions and procedures provided by Oracle	Try minimize writing your own functions and procedures.
2	Use Oracle's Searching and Sorting routines	The built-in routines are highly optimized.

Number	PL/SQL Best Practice	Comments
3	Take advantage of the ways Oracle performs control logic evaluation	If you have multiple conditions that control branching, Oracle evaluates them in the order you provide them. It will not evaluate all the conditions unless it needs to. Order your conditions in a manner that allows Oracle to take short cuts. If there are two conditions, then put the most restrictive condition first.
4	Avoid implicit datatype conversions	Avoid comparing variables that have different datatypes. The time spent on the implicit datatype conversions during each execution could be reclaimed if the datatypes were converted to a consistent set prior to comparisons.
5	Size `VARCHAR2` variables properly	`VARCHAR2(1000)` vs. `VARCHAR2(2000)` If the size is less than 2000, then PL/SQL allocates enough memory to hold the declared length of the variable. But if the size is greater than or equal to 2000, PL/SQL dynamically allocates only enough memory to handle the actual value.
6	Use PL/SQL within SQL statements	There are potential performance gains by including a PL/SQL function as part of a query.
7	Use `DBMS_PROFILER` to identify problems	Capture the profiling statistics to identify the lines in your code that take the most time.
8	Use PL/SQL features for Bulk operations	Avoid row by row operations and use `FORALL` and `BULK COLLECT`.
9	Use `JOIN` methods carefully	Based on the conditions in your query, the available indexes, and available statistics, the optimizer chooses which `JOIN` operation to use. You can influence the optimizer to use a different `JOIN` method.

Abbreviations and Acronyms

APEX - Application Express

RAP - Retail Analytics Platform

RPASCE - Retail Predictive Application Server Cloud Edition

IW - Innovation Workbench

LHS - Left hand side (of expression)

RHS - Right hand side (of expression)

CMF - customer managed fact (rule group property)

Input Data Extensibility

There are additional ways to provide input data to RAP for attributes and measures you want available to RPASCE applications while still leveraging the common foundation input file formats. Some of these extensions are designed specifically for Planning (such as additional fact measure fields on some interfaces) while others are shared across all of RAP (such as the flexible fact tables in RI).

Additional Source for Product Attributes

If you are integrating product attribute data from RMFCS, then you may encounter scenarios where not all product attributes are available in that system and you need to load them separately for Planning purposes only. You have the option to load additional product attributes directly to RAP on the existing foundation interface files for `ATTR.csv` and `PROD_ATTR.csv`, and this data will be merged with the RMFCS foundation data. This is handled using special jobs in the AIF DATA batch schedule listed below:

- SI_W_RTL_PRODUCT_ATTR_DS_MERGE_JOB
- SI_W_RTL_ITEM_GRP1_DS_MERGE_JOB
- SI_W_DOMAIN_MEMBER_DS_TL_MERGE_JOB

These jobs should not be enabled in POM unless you are planning to load data from an additional source, because they will conflict with the normal version of these interfaces (which follows a truncate and load process instead of a merge). When the `MERGE` jobs are turned on, it is assumed that all other `SI_*` type jobs (like `SI_W_RTL_ITEM_GRP1_DS_JOB`) are already disabled, because the core foundation dataset is coming directly from RMFCS and not from flat files. The only exception is that you will also need to keep the `COPY/STG` jobs enabled for the files you are sending in:

- COPY_SI_ATTR_JOB / STG_SI_ATTR_JOB
- COPY_SI_PROD_ATTR_JOB / STG_SI_PROD_ATTR_JOB

The merge functionality is limited to two types of attributes: `ITEMUDA` and `ITEMLIST`. New attributes going to AIF or Planning must be of type `ITEMUDA` and it is the implementer's responsibility to ensure the attributes being added from the secondary source do not conflict with the data coming from RMFCS. The `UDA` attributes will be combined in the same table in RAP and downstream applications like AIF and MFP will have no knowledge of which source provided the data. `ITEMLIST` data from secondary sources is meant specifically for the LPO application in AIF when you are leveraging product groups in that application and do not want to use RMFCS item lists for that purpose.

Additional Source for Foundation Data

Some foundation data interfaces support merging two sources of information together into the same target table. One of the sources must be MFCS while the other may be an external source that generates a CSV file for the nightly batch. This is currently supported for Promotions, Shipments, Sales, and Inventory. These jobs should not be enabled in POM unless you are planning to load data from an additional source, because they will conflict with the normal version of these interfaces (which follows a truncate and load process instead of a merge). When the MERGE jobs are turned on, it is expected that all other `SI_*` type jobs (like `SI_W_RTL_SHIP_DETAILS_DS_JOB`) will be manually disabled, because the core foundation dataset is coming directly from MFCS and not from these jobs.

Secondary Data Source	AIF DATA Jobs to Enable	AIF DATA Jobs to Disable
PROMO_DETAIL.csv	SI_W_RTL_PROMO_IT_LC_DS_MERGE_JOB COPY_SI_PROMO_DETAIL_JOB STG_SI_PROMO_DETAIL_JOB	SI_W_RTL_PROMO_IT_LC_DS_JOB
SHIPMENT_HEAD.csv	SI_W_RTL_SHIP_DETAILS_DS_MERGE_JOB COPY_SI_SHIPMENT_HEAD_JOB STG_SI_SHIPMENT_HEAD_JOB	SI_W_RTL_SHIP_DETAILS_DS_JOB
SHIPMENT_DETAIL.csv	SI_W_RTL_SHIP_IT_LC_DY_FS_MERGE_JOB COPY_SI_SHIPMENT_DETAIL_JOB STG_SI_SHIPMENT_DETAIL_JOB	SI_W_RTL_SHIP_IT_LC_DY_FS_JOB
SALES.csv	SI_W_RTL_SLS_TRX_IT_LC_DY_FS_MERGE_JOB COPY_SI_SALES_JOB STG_SI_SALES_JOB	SI_W_RTL_SLS_TRX_IT_LC_DY_FS_JOB
SALES_PACK.csv	SI_W_RTL_SLSPK_IT_LC_DY_FS_MERGE_JOB COPY_SI_SALES_PACK_JOB STG_SI_SALES_PACK_JOB	SI_W_RTL_SLSPK_IT_LC_DY_FS_JOB
INVENTORY.csv	SI_W_RTL_INV_IT_LC_DY_FS_MERGE_JOB COPY_SI_INVENTORY_JOB STG_SI_INVENTORY_JOB	SI_W_RTL_INV_IT_LC_DY_FS_JOB

The way the data is merged depends on the interface. For `PROMO_DETAIL.csv`, the Pricing CS data always takes priority, and the CSV file data is inserted where it does not match an existing record. This is because the promotion header interface (`W_RTL_PROMO_DS` / `D` tables and `PROMOTION.csv` file) from Pricing CS already follows that logic and this detail-level table needs to match it. You should aim to ensure that there is no overlap between the Pricing CS data and the external CSV file data.

Fact data uses configurable merge logic. For shipment data, it is a configuration option whether you want MFCS data or the CSV files to get first priority when merging. Update the `C_ODI_PARAM_VW` table from the Control Center for parameter `SHIP_SI_MERGE_PRIORITY`. When set to MFCS, the CSV file data will only be inserted if there is no matching record. When set to `EXT` (or any other value), the external file data will overwrite any matching records from MFCS and insert for all other records. Sales and inventory use the same logic based on the value in parameters `SALES_SI_MERGE_PRIORITY` and `INV_SI_MERGE_PRIORITY`.

Additional Source for Data Security

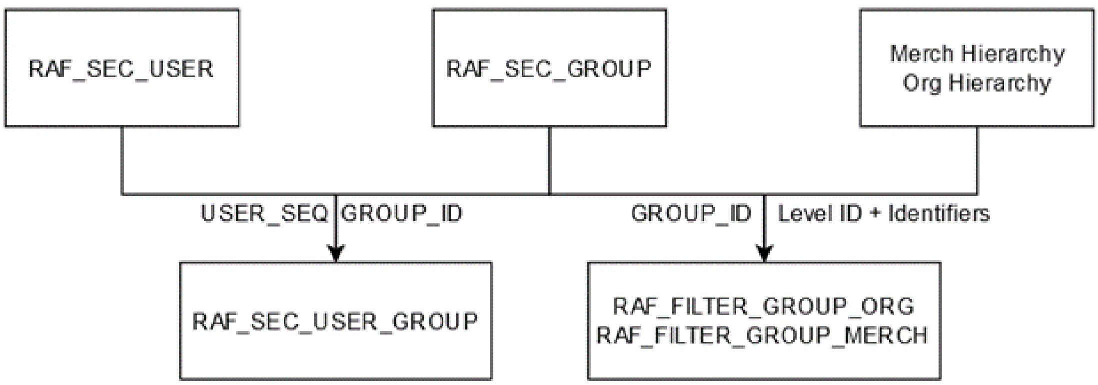
Some applications, such as RI and LPO, use data security configurations to limit what users see in the user interface. These tables are normally populated from MFCS or from a set of flat files such as `RAF_SEC_USER.dat`, as defined in the RAP interfaces guide. If you want to use Innovation Workbench to populate these tables, however, you must leverage an alternate data flow and batch program. The steps to maintain this data from IW are:

1. Create programs or REST APIs that insert data into the following staging tables from IW:

```
RAF_FILTER_GROUP_MERCH_STG
RAF_FILTER_GROUP_ORG_STG
RAF_SEC_USER_STG
RAF_SEC_GROUP_STG
RAF_SEC_USER_GROUP_STG
```

2. From POM, enable and run the adhoc process `RAF_SEC_FILTER_LOAD_ADHOC` in the AIF DATA schedule, which contains just one job named `RAF_SEC_FILTER_LOAD_JOB`. This job truncates the target data warehouse tables (like `RAF_SEC_USER`), then insert the contents from the staging table to the target table.
3. If you are moving the data downstream to LPO or other AIF applications, run the associated data security load jobs for those applications.

The relationship between the internal RAF tables is shown in the diagram below.



> **Note:**
>
> The same set of `RAF_*` tables exist in multiple database schemas, so you must be careful when querying and loading them. When you want to query the tables populated by the AIF DATA job, you must specify `RADM01` as the owner of the table (such as `select * from RADM01.RAF_SEC_USER`). When you want to query the tables owned by AIF APPS, you must specify `RASE01` as the owner.

Additional Sources for Measures

Some of the foundation data file interfaces into RAP have ways to add more measure data than what the out-of-box planning solutions are using.

Custom Sales and Transfer Types

The `SALES.csv` file has transaction data differentiated with a Retail Type (`RTL_TYPE_CODE`) field. By default, this accepts only `R`, `P`, or `C` as the types. You may extend this with a 4th custom type called Other (using type code `O`). To do this, you must first load a custom file into the `W_XACT_TYPE_D` interface to add the additional sales type code (standalone POM jobs are available for this table load). Once that is done, you may include the 4th type code on records in `SALES.csv`. The additional sales type will be exported to PDS in two different ways:

1. The MFP sales interface (`W_PDS_SLS_IT_LC_WK_A`) has a set of fields for Total Sales, which will be inclusive of Other sales. This allows you to have the default measures for Reg, Pro, and Clr sales and custom non-GA measures for Total Sales (which will not be equal to R+P+C sales). You could use total sales measures minus the other types to arrive at values specifically for Other sales or any other combination of retail types.
2. The IPOCS-Demand Forecasting sales interface (`W_PDS_GRS_SLS_IT_LC_WK_A`) will maintain the separate rows for other sales on the output since that interface has the retail type code on it directly. You may define custom measures to load the Other sales into IPOCS-Demand Forecasting.

This process also works for transfer measures coming on the `TRANSFER.csv` file. The transfer type code on this file also comes from the `W_XACT_TYPE_D` table and can be extended with additional type codes to create more measure data in PDS. If you add additional type codes, update your `interface.cfg` to import the new records into custom measures that are filtered by the same type code.

Custom Fact Measures

The following files have been extended with 20 generic numerical fields on the end of the files:

- SALES.csv
- MARKDOWN.csv
- INVENTORY.csv
- ORDER_DETAIL.csv
- DEAL_INCOME.csv
- RECEIPT.csv

These flex fields will be loaded exactly as-is with no conversions or transformations, except to aggregate them from the input data to the base intersection of item/location/fiscal week. All flex fields use SUM as the aggregation method. If a flex field in the output PDS table has a prefix, like `PO_FLEX1_NUM_VALUE`, it means it is additionally splitting the data by that type, such as PO vs. Transfer receipt measures. If there is no prefix, then the flex measure is summed only to item/location/week level. These fields provide a way to send additional custom measures to Planning applications when the source of the measures is the same as your basic foundation data. Refer to the *RAP Data Interfaces Guide* in My Oracle Support (Doc ID 2539848.1) for the complete file specifications.

Additional Custom Fact Data

Most implementations have greater fact and measure requirements than the default solution interfaces provide. While it is possible to load additional data directly into the RPASCE PDS, there is the option to send some of this data into the Retail Insights data model alongside your foundation data files. This makes it available for many more use cases within RAP, such as reporting, data visualization and extensions in IW.

There are four flexible fact interfaces (referred to as Flex Facts) in the Retail Insights data warehouse that can also be used as general-purpose data interfaces in RAP (even if you do not subscribe to RI itself). These interfaces have `W_RTL_FLEXFACT` at the start of the file and table names. For details on how to configure and use these interfaces, refer to the *Retail Insights Implementation Guide* chapter on “Planning and Flex Fact Configurations”.

Extensibility Example – Product Hierarchy

A common scenario for extensibility is the need to add more product hierarchy levels to some solutions like MFP beyond what the base application supports. It is possible to define additional custom hierarchy levels in various RAP solutions following the general workflow below. For this example, we assume a new product level named "Sub-Category" will be added. This level will be placed between the Department and Class levels within the main hierarchy in AIF and RPAS applications.

Input File Changes

To add an additional product hierarchy level into the RAP foundation data files, you must leverage the `FLEX` fields available on either `PRODUCT.csv` or `PRODUCT_ALT.csv` files. Either file may be used, as the columns are merged together into one table before sending it downstream. For this example, we will use `PRODUCT_ALT.csv` columns `FLEX1_CHAR_VALUE` and `FLEX2_CHAR_VALUE`. The `FLEX1` values are the unique identifiers for the hierarchy positions, while the `FLEX2` values are the description for display in user interfaces. The file is generated at the item level the same way as `PRODUCT.csv` and must follow the same rules for hierarchy construction (IDs must be unique within the level, you must not have multiple parents for the same children in the level below this one, and so on). Example rows from the file may be:

```
ITEM,FLEX1_CHAR_VALUE,FLEX2_CHAR_VALUE
30018,100101,WOMEN'S CLOTHING
30019,100101,WOMEN'S CLOTHING
51963371,100103,WOMEN'S INSPIRATION
1101247,100104,WOMEN'S FAST FASHION
```

Once you have generated this data for all items in the hierarchy, then you will load it into the platform following the Initialize Dimensions process in Data Loads and Initial Batch Processing. The following jobs in the `RI_DIM_INITIAL_ADHOC` process are used to load this file:

- COPY_SI_PRODUCT_ALT_JOB
- STG_SI_PRODUCT_ALT_JOB
- SI_W_PRODUCT_FLEX_DS_JOB
- W_PRODUCT_FLEX_D_JOB

You should already have loaded a `PRODUCT.csv` file at this stage, or you should load it at the same time as the `PRODUCT_ALT.csv` file, so that the full product hierarchy is available in the data warehouse. Once loaded, the data for the alternate levels will be available in the `W_PRODUCT_FLEX_D` table for review. At this stage, the data is only available in the data warehouse table; it has not been configured for use in any other solution.

AI Foundation Setup

To see the additional hierarchy level in AI Foundation applications, you must create an alternate product hierarchy that includes both the new level and all other levels from your product hierarchy that you wish to use.

The first step in defining the alternate product hierarchy in AIF is setting up the configuration tables `RSE_ALT_HIER_TYPE_STG` and `RSE_ALT_HIER_LEVEL_STG`. These tables are updated from the Manage System Configurations screen in the Control & Tactical Center. For this example, the data you create may look like the following:

Table 9-11 RSE_ALT_HIER_LEVEL_STG Sample

HIER_TYPE_NAME	HIER_LEVEL_ID	DESCR	ID_SRC_COLUMN_NAME	ID_SRC_OBJ_NAME	NAME_SRC_COLUMN_NAME	NAME_SRC_OBJ_NAME	UI_DESCR
PROD_SUBCAT	1	CMP	TOP_PRODCAT_ID	W_PRODUCT_DTS	TOP_PRODCAT_DESC	W_PRODUCT_DTS	Company
PROD_SUBCAT	2	DIV	LVL8_PRODCAT_ID	W_PRODUCT_DTS	LVL8_PRODCAT_DESC	W_PRODUCT_DTS	Division
PROD_SUBCAT	3	GRP	LVL7_PRODCAT_ID	W_PRODUCT_DTS	LVL7_PRODCAT_DESC	W_PRODUCT_DTS	Group
PROD_SUBCAT	4	DEPT	LVL6_PRODCAT_ID	W_PRODUCT_DTS	LVL6_PRODCAT_DESC	W_PRODUCT_DTS	Department
PROD_SUBCAT	5	SUBCAT	FLEX1_CHAR_VALUE	W_PRODUCT_ALT_DTS	FLEX2_CHAR_VALUE	W_PRODUCT_ALT_DTS	Sub Category
PROD_SUBCAT	6	CLS	LVL5_PRODCAT_ID	W_PRODUCT_DTS	LVL5_PRODCAT_DESC	W_PRODUCT_DTS	Class
PROD_SUBCAT	7	SBC	LVL4_PRODCAT_ID	W_PRODUCT_DTS	LVL4_PRODCAT_DESC	W_PRODUCT_DTS	Sub Class
PROD_SUBCAT	8	SKU	ITEM	W_PRODUCT_DTS	ITEM_DESC	W_PRODUCT_DTS	Stock Keeping Unit

Table 9-12 RSE_ALT_HIER_TYPE_STG Sample

DELETE_FLG	DESCR	NAME	OBJ_TYPE_NAME
N	Product Hierarchy with Sub-Category	PROD_SUBCAT	PRODUCT

The configurations specified in this example show how to refer to the default hierarchies (which are loaded through the staging table `W_PRODUCT_DTS`) and the alternate hierarchies (loaded through the table `W_PRODUCT_ALT_DTS`). When referring to a default hierarchy level, you should use the parameters shown here for all the `SRC` fields. You can modify the `HIER_LEVEL_ID` to change the placement of the levels within the structure; however the standard hierarchy rules must still pass after reorganizing them (for example, you cannot place `DEPT` below `CLS` because then the same child node may have multiple parent nodes).

After your configuration is finalized, you may generate the alternate hierarchy in AIF using `RSE_MASTER_ADHOC_JOB` with the `-X` flag. This will only load the alternate hierarchy; it assumes you have also loaded the main hierarchy using the `-p` flag, or you are loading both of them together using `-pX`. For nightly batch job details, refer to the *AI Foundation Implementation Guide*, section “Building Alternate Hierarchy in AIF”.

It is also necessary to update `RSE_CONFIG` options to use the new hierarchy. For example, to use the hierarchy in LPO, change the `PMO_PROD_HIER_TYPE` parameter to the ID for the new hierarchy. You can find the ID for the hierarchy in table `RSE_ HIER_TYPE` column `ID`, which is viewable in Manage System Configurations. Custom hierarchies will have `ALT_FLG=Y` in their rows of the table.

If you will use the alternate hierarchy in forecast generation for Planning, then the rest of the data aggregation and forecasting processes are the same, whether you are using the standard product hierarchy or the alternate one. You will follow all steps outlined in the *AI Foundation Implementation Guide* sections for “Forecast Configuration for MFP and AP” and “Forecast Configuration for IPO-DF and AIF” as needed. A summary of those steps are:

1. Set up the configuration to use your alternate hierarchy
2. Create your run types and select your desired intersections, which can include the new alternate hierarchy levels as the forecast level
3. Perform aggregation, estimation, and forecasting processes following the usual steps in the AIF guides
4. Run the ad hoc jobs from POM to export the forecast results to Planning, such as `RSE_MFP_FCST_EXPORT_ADHOC_JOB`

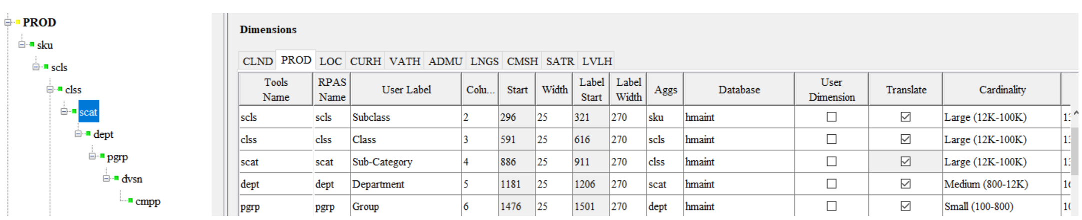
If you generate a forecast using the custom level, then the export to PDS will appear for that level description as defined in `RSE_ALT_HIER_LEVEL_STG.DESCR`. In this example, you may generate a forecast at the SUBCAT / AREA / Fiscal Week levels for use in MFP. These are the level names that will appear in the forecast export and must be configured for use in MFP.

Planning Data Store Setup

Planning applications such as MFP and AP can also leverage the same additional hierarchy levels provided on the foundation input files. The first step is to export the hierarchies from the data warehouse to PDS. This can be done using the same set of ad hoc jobs in the AIF DATA schedule in POM, as described in Sending Data to Planning. The flex fields from `W_PRODUCT_FLEX_D` will be written to the same PDS staging table, `W_PDS_PRODUCT_D`.

Once it reaches the staging table in the RDX schema, the same can be interfaced to PDS hierarchies by making changes to `interface.cfg`. Follow the steps below for integrating the new dimension into PDS for the Product Hierarchy, which includes changes to `interface.cfg` for importing the dimension and to export and import AIF data at the new dimension level.

- Update the configuration for either GA (template activated) or non-GA (template de-activated) to include the new dimension in the hierarchy structure. In the example below, say ‘Sub-Category’ was added as dimension ‘scat’ between Class and Department.



- Update the `interface.cfg` to interface the newly added dimension from the corresponding mapped column from RDX.

  In the below example we added entries for `HDM50` and `HDL50` to map the dimension position and label for the dimension from the RDX staging table. If you are using the GA template or if you are not using a template but starting from GA configuration, use numbers starting from `50` for new dimensions. If it is a fully custom configuration, you may use any numbering.

```
# Hierarchy Importer Mappings for Product Hierarchy

W_PDS_PRODUCT_D:PDS:HDM01:SKU:ITEM:
```

```
W_PDS_PRODUCT_D:PDS:HDM04:SCLS:SUBCLASS_ID:
W_PDS_PRODUCT_D:PDS:HDM05:CLSS:CLASS_ID:
W_PDS_PRODUCT_D:PDS:HDM06:DEPT:DEPT:
W_PDS_PRODUCT_D:PDS:HDM07:PGRP:GROUP_NO:
W_PDS_PRODUCT_D:PDS:HDM08:DVSN:DIVISION:
W_PDS_PRODUCT_D:PDS:HDM09:CMPP:COMPANY:
W_PDS_PRODUCT_D:PDS:HDM50:SCAT:FLEX1_CHAR_VALUE:
W_PDS_PRODUCT_D:PDS:HDL01::ITEM_DESC:
W_PDS_PRODUCT_D:PDS:HDL04::SUB_NAME:
W_PDS_PRODUCT_D:PDS:HDL05::CLASS_NAME:
W_PDS_PRODUCT_D:PDS:HDL06::DEPT_NAME:
W_PDS_PRODUCT_D:PDS:HDL07::GROUP_NAME:
W_PDS_PRODUCT_D:PDS:HDL08::DIV_NAME:
W_PDS_PRODUCT_D:PDS:HDL09::CO_NAME:
W_PDS_PRODUCT_D:PDS:HDL50::FLEX2_CHAR_VALUE:
```

> **Note:**
>
> If using GA template with extensibility, you also need to add `custom_add` as the last column for newly added columns.

```
W_PDS_PRODUCT_D:PDS:HDM50:SCAT:FLEX1_CHAR_VALUE::custom_add
…
W_PDS_PRODUCT_D:PDS:HDL50::FLEX2_CHAR_VALUE::custom_add
```

- To export plans generated at the new level to AIF to use in forecast generation, then create plans at the new level and export the plans defined at that level to AIF. Assuming the intersection of the plans are new dimension level, ensure the product dimension (`DIM02` in the example below) (which is mapped to `PROD_KEY`)is set to `SCAT` to identify the product intersection of data in PDS as Sub-Category. For AIF to understand the prod level as Sub-Category, set the `PROD_LEVEL` value to `SUBCAT`, as defined in the AIF alternate hierarchy setup.

```
MFP_PLAN1_EXP:MPOP:DIM01:WEEK:CLND_KEY:
MFP_PLAN1_EXP:MPOP:DIM02:SCAT:PROD_KEY:
MFP_PLAN1_EXP:MPOP:DIM03:CHNC:LOC_KEY:MFP_PLAN1_EXP:MPOP:DATA::CLND_LEVEL:W
EEK
MFP_PLAN1_EXP:MPOP:DATA::PROD_LEVEL:SUBCAT
MFP_PLAN1_EXP:MPOP:DATA::LOC_LEVEL:AREA
…
MFP_PLAN1_EXP:MPOP:DATA:MFP_MPOPLDOWD:CAL_DATE:
MFP_PLAN1_EXP:MPOP:DATA:MFP_MPOPSLSU:SLS_QTY:
MFP_PLAN1_EXP:MPOP:DATA:MFP_MPOPSLSR:SLS_RTL_AMT:
```

> **Note:**
>
> Some export tables to AIF may not have `PROD_LEVEL` or `PROD_HIER_LEVEL` defined. If not they are not present, then that specific interface table is only meant for pre-defined product levels and you cannot change it.

- If AIF is generating the forecast at the new 'SUBCAT' level and exporting the forecast data, then the same can be pulled into MFP using the following updates to the forecast interface. Assuming the new forecast measures are defined at the Sub-Category level instead of existing Subclass level in GA, then below are the changes needed. Update the dimension for product to `SCAT` to specify the intersection for import measures as identified by PDS and also set the filter criteria for imported data in `PROD_HIER_LEVEL` to `SUBCAT` as identified by AIF hierarchy setup.

```
RSE_FCST_DEMAND_EXP:MPP:DIM01:WEEK:FCST_DATE_FROM:
RSE_FCST_DEMAND_EXP:MPP:DIM02:SCAT:PROD_EXT_KEY:
RSE_FCST_DEMAND_EXP:MPP:DIM03:CHNC:LOC_EXT_KEY:
RSE_FCST_DEMAND_EXP:MPP:DATA:MFP_MPWPDMDP1U:REG_PR_SLS_QTY:
RSE_FCST_DEMAND_EXP:MPP:DATA:MFP_MPWPDMDP1R:REG_PR_SLS_AMT:
…
RSE_FCST_DEMAND_EXP:MPP:FILTER::CAL_HIER_LEVEL:Fiscal Week
RSE_FCST_DEMAND_EXP:MPP:FILTER::PROD_HIER_LEVEL:SUBCAT
RSE_FCST_DEMAND_EXP:MPP:FILTER::LOC_HIER_LEVEL:CHANNEL
RSE_FCST_DEMAND_EXP:MPP:FILTER::CUSTSEG_EXT_KEY:
RSE_FCST_DEMAND_EXP:MPP:FILTER::FCST_TYPE:NPI
```

> **Note:**
>
> Some import tables from AIF may not have `PROD_LEVEL` or `PROD_HIER_LEVEL` defined. If they are not present, then that specific interface table is only meant for pre-defined product levels and you cannot change it.

In-Season Forecast Setup

All the prior steps describe the setup needed to get the first round of plans and forecasts generated, which for MFP is the pre-season workflow. If you also plan to use forecasts for in-season planning in MFP then you must configure the flow of approved plans from MFP back through the data warehouse for AIF to consume. This data flow starts from the MFP export tables (such as `MFP_PLAN1_EXP`) and passes through the data warehouse plan tables (such as `W_RTL_PLAN1_PROD1_LC1_T1_F`).

To configure these interfaces, use the parameters on `C_ODI_PARAM_VW` in the Manage System Configurations screen. Our plan data at the levels of `SUBCAT` / `AREA` / `WEEK` will need this set of parameters:

Param Name	Param Value
RI_PLAN1_CAL_LEVEL	WEEK
RI_PLAN1_ORG_LEVEL	AREA
RI_PLAN1_PROD_LEVEL	FLEX1
RI_PLAN1_SUPP_LEVEL	ALL
RI_PLAN1_ATTR_LEVEL	ALL

The product level of `FLEX1` correlates with the column in the `W_PRODUCT_ALT_DTS` table that you used to load the alternate hierarchy level in the very beginning of the process, and matches the field mapped during AIF alternate hierarchy setup.

To integrate the data from MFP to the data warehouse, the jobs in the AIF DATA schedule in POM that are used are:

- W_RTL_PLAN1_PROD1_LC1_T1_FS_SDE_JOB
- W_RTL_PLAN1_PROD1_LC1_T1_F_JOB

These jobs are included in the AIF DATA nightly schedule and can also be found in the ad hoc process `LOAD_PLANNING1_DATA_ADHOC`. This process populates the table `W_RTL_PLAN1_PROD1_LC1_T1_F`, which can then be loaded to the AIF forecasting module using the AIF APPS schedule job `RSE_FCST_SALES_PLAN_LOAD_JOB`. This job populates the table `RSE_FCST_SALES_PLAN_DTL` which is used in generating plan-influenced forecasts.

A

Legacy Foundation File Reference

The following table provides a cross-reference for legacy application input files and the Retail Analytics and Planning files that replace them. This list covers foundation data flows which span multiple applications, such as MFP and RI. Other foundation files exist which do not replace multiple application files; those are specified in the Interfaces Guide in My Oracle Support.

File Group	File Type	Legacy Planning Files	Legacy RI/AI Foundation Files	RAP Files
Product	Dimension	prod.csv.dat	W_PRODUCT_DS.dat W_PRODUCT_DS_TL.dat W_PROD_CAT_DHS.dat W_DOMAIN_MEMBER_DS_TL.dat W_RTL_PRODUCT_BRAND_DS.dat W_RTL_PRODUCT_BRAND_DS_TL.dat W_RTL_IT_SUPPLIER_DS.dat W_PARTY_ORG_DS.dat W_RTL_PRODUCT_IMAGE_DS.dat W_PRODUCT_ATTR_DS.dat W_RTL_ITEM_GRP1_DS.dat	PRODUCT.csv
Organization	Dimension	loc.csv.dat stor_metrics.csv.ovr	W_INT_ORG_DS.dat W_INT_ORG_DS_TL.dat W_INT_ORG_DHS.dat W_DOMAIN_MEMBER_DS_TL.dat (for RTL_ORG) W_RTL_CHANNEL_DS.dat W_INT_ORG_ATTR_DS.dat	ORGANIZATION.csv
Calendar	Dimension	clnd.csv.dat	W_MCAL_PERIOD_DS.dat	CALENDAR.csv
Exchange Rates	Dimension	curh.csv.dat curr.csv.ovr	W_EXCH_RATE_GS.dat	EXCH_RATE.csv
Attributes	Dimension	patr.csv.dat patv.csv.ovr	W_RTL_PRODUCT_ATTR_DS.dat W_RTL_PRODUCT_ATTR_DS_TL.dat W_DOMAIN_MEMBER_DS_TL.dat (for Diffs) W_RTL_PRODUCT_COLOR_DS.dat	ATTR.csv

File Group	File Type	Legacy Planning Files	Legacy RI/AI Foundation Files	RAP Files
Diff Groups	Dimension	sizh.hdr.csv.dat	W_RTL_DIFF_GRP1_DS.dat W_RTL_DIFF_GRP1_DS_TL.dat	DIFF_GROUP.csv
Product Attribute Assignments	Dimension	prdatt.csv.ovr	W_RTL_ITEM_GRP1_DS.dat	PROD_ATTR.csv
Sales	Fact	rsal.csv.ovr psal.csv.ovr csal.csv.ovr nsls.csv.ovr rtn.csv.ovr	W_RTL_SLS_TRX_IT_LC_DY_FS.dat W_RTL_SLSPK_IT_LC_DY_FS.dat	SALES.csv SALES_PACK.csv
Inventory	Fact	eop.csv.ovr eopx.csv.ovr wsal.csv.ovr	W_RTL_INV_IT_LC_DY_FS.dat	INVENTORY.csv
Markdown	Fact	mkd.csv.ovr	W_RTL_MKDN_IT_LC_DY_FS.dat	MARKDOWN.csv
On Order	Fact	oo.csv.ovr	W_RTL_PO_DETAILS_DS.dat W_RTL_PO_ONDORD_IT_LC_DY_FS.dat	ORDER_HEAD.csv ORDER_DETAIL.csv
PO Receipts	Fact	rcpt.csv.ovr	W_RTL_INVRC_IT_LC_DY_FS.dat	RECEIPT.csv
Transfers	Fact	tranx.csv.ovr	W_RTL_INVTSF_IT_LC_DY_FS.dat	TRANSFER.csv
Adjustments	Fact	tran.csv.ovr	W_RTL_INVADJ_IT_LC_DY_FS.dat W_REASON_DS.dat	ADJUSTMENT.csv REASON.csv
RTVs	Fact	tran.csv.ovr	W_RTL_INVRTV_IT_LC_DY_FS.dat	RTV.csv
Costs	Fact	slsprc.csv.ovr	W_RTL_BCOST_IT_LC_DY_FS.dat W_RTL_NCOST_IT_LC_DY_FS.dat	COST.csv
Prices	Fact	slsprc.csv.ovr	W_RTL_PRICE_IT_LC_DY_FS.dat	PRICE.csv
W/F Sales and Fees	Fact	tran.csv.ovr	W_RTL_SLSWF_IT_LC_DY_FS.dat	SALES_WF.csv
Vendor Funds (TC 6/7)	Fact	tran.csv.ovr	W_RTL_DEALINC_IT_LC_DY_FS.dat	DEAL_INCOME.csv
Reclass In/Out (TC 34/36)	Fact	tran.csv.ovr	W_RTL_INVRECLASS_IT_LC_DY_FS.dat	INV_RECLASS.csv
Intercompany Margin (TC 39)	Fact	tran.csv.ovr	W_RTL_ICM_IT_LC_DY_FS.dat	IC_MARGIN.csv

B

Context File Table Reference

The following table maps CSV data files to internal tables for the purpose of creating Context Files. The first parameter on the Context file is a `TABLE` property containing the table name into which the CSV data will be loaded. For legacy context file usage, the name of the context file itself should match the internal table name.

File Type	Filenames	Internal Tables
Product	PRODUCT.csv	W_PRODUCT_DTS
Product Alternates	PRODUCT_ALT.csv	W_PRODUCT_ALT_DTS
Organization	ORGANIZATION.csv	W_INT_ORG_DTS
Organization Alternates	ORGANIZATION_ALT.csv	W_ORGANIZATION_ALT_DTS
Calendar	CALENDAR.csv	W_MCAL_PERIODS_DTS
Exchange Rates	EXCH_RATE.csv	W_EXCH_RATE_DTS
Attributes	ATTR.csv	W_ATTR_DTS
Diff Groups	DIFF_GROUP.csv	W_DIFF_GROUP_DTS
Product Attribute Assignments	PROD_ATTR.csv	W_PRODUCT_ATTR_DTS
Employee	EMPLOYEE.csv	W_EMPLOYEE_DTS
Application Codes	CODES.csv	W_RTL_CODE_DTS
Pack Items	PROD_PACK.csv	W_RTL_ITEM_GRP2_DTS
Promotions	PROMOTION.csv	W_RTL_PROMO_EXT_DTS
Supplier	SUPPLIER.csv	W_SUPPLIER_DTS
Item Loc Attributes	PROD_LOC_ATTR.csv	W_PROD_LOC_ATTR_DTS
Replenishment Attributes	PROD_LOC_REPL.csv	W_INVENTORY_PRODUCT_ATTR_DTS
Season Phase	SEASON.csv	W_RTL_SEASON_PHASE_DTS
Season Phase Item Mapping	PROD_SEASON.csv	W_RTL_SEASON_PHASE_IT_DTS
Comp Stores	STORE_COMP.csv	W_RTL_LOC_COMP_MTX_DTS
Item Deletes	PROD_DELETE.csv	W_RTL_ITEM_DEL_TMPS
Item Loc Deletes	PROD_LOC_DELETE.csv	W_RTL_IT_LC_DEL_TMPS
Sales	SALES.csv	W_RTL_SLS_TRX_IT_LC_DY_FTS
Sales Pack	SALES_PACK.csv	W_RTL_SLSPK_IT_LC_DY_FTS
Inventory	INVENTORY.csv	W_RTL_INV_IT_LC_DY_FTS
Inventory Packs	INVENTORY_PACK.csv	W_RTL_INVPK_IT_LC_DY_FTS
Markdown	MARKDOWN.csv	W_MARKDOWN_FTS
On Order	ORDER_HEAD.csv	W_ORDER_HEAD_FTS
On Order Detail	ORDER_DETAIL.csv	W_ORDER_DETAIL_FTS
PO Receipts	RECEIPT.csv	W_RECEIPT_FTS

File Type	Filenames	Internal Tables
Transfers	TRANSFER.csv	W_RTL_INVTSF_IT_LC_DY_FTS
Adjustments	ADJUSTMENT.csv	W_ADJUSTMENT_FTS
RTVs	RTV.csv	W_RTL_INVRTV_IT_LC_DY_FTS
Costs	COST.csv	W_COST_FTS
Prices	PRICE.csv	W_RTL_PRICE_IT_LC_DY_FTS
W/F Sales and Fees	SALES_WF.csv	W_RTL_SLSWF_IT_LC_DY_FTS
Vendor Funds	DEAL_INCOME.csv	W_RTL_DEALINC_IT_LC_DY_FTS
Reclass In/Out	INV_RECLASS.csv	W_RTL_INVRECLASS_IT_LC_DY_FTS
Intercompany Margin	IC_MARGIN.csv	W_RTL_ICM_IT_LC_DY_FTS

C

Sample Public File Transfer Script for Planning Apps

This example provides an example of how file transfers can be implemented through a shell script. It requires: bash, curl and jq.

```
#!/bin/bash

# Sample Public FTS script

### EDIT HERE to reflect your environment

BASE_URL="https://__YOUR_TENANT_BASE_URL__"
TENANT="__YOUR-TENANT_ID__"
IDCS_URL="https://_YOUR__IDCS__URL__/oauth2/v1/token"
IDCS_CLIENTID="__YOUR_CLIENT_APPID__"
IDCS_CLIENTSECRET="__YOUR_CLIENT_SECRET___"
IDCS_SCOPE="rgbu:rpas:psraf-__YOUR_SCOPE__"

### FINISHED

clientToken() {
    curl  -sX POST "${IDCS_URL}" \
          --header "Authorization: Basic ${IDCS_AUTH}" \
          --header "Content-Type: application/x-www-form-urlencoded" \
          --data-urlencode "grant_type=client_credentials" \
          --data-urlencode "scope=${IDCS_SCOPE}"  | jq -r .access_token
}

ping() {
    echo "Pinging"
    curl -sfX GET "${BASE_URL}/${TENANT}/RetailAppsReSTServices/services/
private/FTSWrapper/ping" \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" | jq
}


listPrefixes() {
    echo "Listing storage prefixes"
    curl -sfX GET "${BASE_URL}/${TENANT}/RetailAppsReSTServices/services/
private/FTSWrapper/listprefixes" \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" | jq
}
```

```
listFiles() {
    echo "Listing files for ${1}"
    curl -sfX GET "${BASE_URL}/${TENANT}/RetailAppsReSTServices/services/
private/FTSWrapper/listfiles?prefix=${1}" \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" | jq
}

deleteFiles() {
    echo "Deleting files"
    json=$(fileCollection $@)
    curl --show-error -sfX DELETE "${BASE_URL}/${TENANT}/
RetailAppsReSTServices/services/private/FTSWrapper/delete" \
         --header 'content-type: application/json' \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" \
         -d "${json}" | jq
}

fileMover() {
    movement="${1}"
    shift

    json=$(fileCollection $@)
    requestPAR "${movement}" "${json}"
}

fileCollection() {
    local json="{  \"listOfFiles\": [ __FILES__ ] }"

    sp="${1}"
    shift

    while (( "${#}" )); do
        list="${list} { \"storagePrefix\": \"${sp}\", \"fileName\":
\"${1}\" }"
        if [[ ${#} -gt "1" ]]; then
            list="${list},"
        fi
        shift
    done

    echo "${json/__FILES__/${list}}"
}

requestPAR() {
    use="${1}"
    echo "Requesting PARs for ${use}"
    pars="$(curl --show-error -sfX POST "${BASE_URL}/${TENANT}/
RetailAppsReSTServices/services/private/FTSWrapper/${use}" \
         --header 'content-type: application/json' \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" \
```

```
         -d "${2}")"

    if [[ "$?" -ne "0" ]]; then
        echo "Error retreiving PAR, check files specified."
        echo "${pars}"
        exit 1
    fi

    list=$(jq -r ".parList[]|[.name, .accessUri]|@csv" <<< "${pars}")

    while IFS='' read -r line; do
        fn=$(echo ${line}|cut -d\, -f1|tr -d \")
        par=$(echo ${line}|cut -d\, -f2|tr -d \")

        if [[ ${use} == "upload" ]]; then
            echo "Uploading ${fn} to ${par}"
            curl -o log -X PUT --data-binary "@${fn}" "${par}"
        else
            echo "Downloading ${fn} from ${par}"
            curl -o ${fn} -X GET "${par}"
        fi
    done <<< "${list}"
}


#Entry point
IDCS_AUTH=$(echo -n ${IDCS_CLIENTID}:${IDCS_CLIENTSECRET} | base64 -w0)
CLIENT_TOKEN=$(clientToken)

case "${1}" in
    ping)
        ping
        ;;
    listprefixes)
        shift
        listPrefixes
        ;;
    listfiles)
        shift
        listFiles ${@}
        ;;
    deletefiles)
        shift
        deleteFiles ${@}
        ;;
    uploadfiles)
        shift
        fileMover upload ${@}
        ;;
    downloadfiles)
        shift
        fileMover download ${@}
        ;;
    *)
        echo "Usage: $0"
        echo " ping                                      : test service
```

```
functionality"
        echo " listprefixes                               : list registered
prefixes"
        echo " listfiles [prefix]                         : list files within
a prefix"
        echo " deletefiles [prefix] [file1] [file2] ...   : delete files with
this prefix"
        echo " uploadfiles [prefix] [file1] [file2] ...   : upload files with
this prefix"
        echo " downloadfiles [prefix] [file1] [file2] ... : download files
with this prefix"
        echo
        exit 0
        ;;
esac
```

D

Sample Public File Transfer Script for RI and AIF

```
#!/bin/bash

# Sample Public FTS script

### EDIT HERE to reflect your environment

BASE_URL="https://__YOUR_TENANT_BASE_URL__"
TENANT="__YOUR-TENANT_ID__"
IDCS_URL="https://_YOUR__IDCS__URL__/oauth2/v1/token"
IDCS_CLIENTID="__YOUR_CLIENT_APPID__"
IDCS_CLIENTSECRET="__YOUR_CLIENT_SECRET___"
IDCS_SCOPE="rgbu:rsp:psraf-__YOUR_SCOPE__"

### FINISHED

clientToken() {
 curl  -sX POST "${IDCS_URL}" \
          --header "Authorization: Basic ${IDCS_AUTH}" \
          --header "Content-Type: application/x-www-form-urlencoded" \
          --data-urlencode "grant_type=client_credentials" \
          --data-urlencode "scope=${IDCS_SCOPE}"  | jq -r .access_token
}

ping() {
    echo "Pinging"
    curl -sfX GET "${BASE_URL}/${TENANT}/RIRetailAppsPlatformServices/
services/private/FTSWrapper/ping" \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" | jq
}


listPrefixes() {
    echo "Listing storage prefixes"
    curl -sfX GET "${BASE_URL}/${TENANT}/RIRetailAppsPlatformServices/
services/private/FTSWrapper/listprefixes" \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" | jq
}


listFiles() {
    echo "Listing files for ${1}"
    curl -sfX GET "${BASE_URL}/${TENANT}/RIRetailAppsPlatformServices/
```

```
services/private/FTSWrapper/listfiles?prefix=${1}" \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" | jq
}

deleteFiles() {
    echo "Deleting files"
    json=$(fileCollection $@)
    curl --show-error -sfX DELETE "${BASE_URL}/${TENANT}/
RIRetailAppsPlatformServices/services/private/FTSWrapper/delete" \
         --header 'content-type: application/json' \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" \
         -d "${json}" | jq
}

fileMover() {
    movement="${1}"
    shift

    json=$(fileCollection $@)
    requestPAR "${movement}" "${json}"
}

fileCollection() {
    local json="{  \"listOfFiles\": [ __FILES__ ] }"

    sp="${1}"
    shift

    while (( "${#}" )); do
        list="${list} { \"storagePrefix\": \"${sp}\", \"fileName\":
\"${1}\" }"
        if [[ ${#} -gt "1" ]]; then
            list="${list},"
        fi
        shift
    done

    echo "${json/__FILES__/${list}}"
}

requestPAR() {
    use="${1}"
    echo "Requesting PARs for ${use}"
    pars="$(curl --show-error -sfX POST "${BASE_URL}/${TENANT}/
RIRetailAppsPlatformServices/services/private/FTSWrapper/${use}" \
         --header 'content-type: application/json' \
         --header 'Accept: application/json' \
         --header 'Accept-Language: en' \
         --header "Authorization: Bearer ${CLIENT_TOKEN}" \
         -d "${2}")"

    if [[ "$?" -ne "0" ]]; then
```

```
        echo "Error retreiving PAR, check files specified."
        echo "${pars}"
        exit 1
    fi

    list=$(jq -r ".parList[]|[.name, .accessUri]|@csv" <<< "${pars}")

    while IFS='' read -r line; do
        fn=$(echo ${line}|cut -d\, -f1|tr -d \")
        par=$(echo ${line}|cut -d\, -f2|tr -d \")

        if [[ ${use} == "upload" ]]; then
            echo "Uploading ${fn} to ${par}"
            curl -o log -X PUT --data-binary "@${fn}" "${par}"
        else
            echo "Downloading ${fn} from ${par}"
            curl -o ${fn} -X GET "${par}"
        fi
    done <<< "${list}"
}


#Entry point
IDCS_AUTH=$(echo -n ${IDCS_CLIENTID}:${IDCS_CLIENTSECRET} | base64 -w0)
CLIENT_TOKEN=$(clientToken)

case "${1}" in
    ping)
        ping
        ;;
    listprefixes)
        shift
        listPrefixes
        ;;
    listfiles)
        shift
        listFiles ${@}
        ;;
    deletefiles)
        shift
        deleteFiles ${@}
        ;;
    uploadfiles)
        shift
        fileMover upload ${@}
        ;;
    downloadfiles)
        shift
        fileMover download ${@}
        ;;
    *)
        echo "Usage: $0"
        echo " ping                                      : test service
functionality"
        echo " listprefixes                              : list registered
prefixes"
```

```
        echo " listfiles [prefix]                              : list files within
a prefix"
        echo " deletefiles [prefix] [file1] [file2] ...    : delete files with
this prefix"
        echo " uploadfiles [prefix] [file1] [file2] ...    : upload files with
this prefix"
        echo " downloadfiles [prefix] [file1] [file2] ... : download files
with this prefix"
        echo
        exit 0
        ;;
esac
```

E

Sample Validation SQLs

This set of sample SQL commands provides scripts to run using APEX which can help validate your initial dimension and fact loads, especially if it is the first time loading the data and quality is unknown. Do not load data into the platform without performing some level of validation on it first, as this will greatly reduce the time spent reworking and reloading data.

```
------------------------------------------------
-- Checks for CALENDAR.csv file load
------------------------------------------------
-- Verify initial calendar data before staging it further, row counts should
match data file
SELECT * FROM W_MCAL_PERIOD_DTS

-- Check total counts, all counts should be same. This can indirectly check
for nulls in required columns.
SELECT
count(*),count(MCAL_CAL_ID),count(MCAL_PERIOD_TYPE),count(MCAL_PERIOD_NAME),
count(MCAL_PERIOD),count(MCAL_PERIOD_ST_DT),count(MCAL_PERIOD_END_DT),count(MC
AL_QTR),
count(MCAL_YEAR),count(MCAL_QTR_START_DT),count(MCAL_QTR_END_DT),count(MCAL_YE
AR_START_DT),
count(MCAL_YEAR_END_DT) FROM W_MCAL_PERIOD_DTS

-- This should not return any rows
SELECT * FROM W_MCAL_PERIOD_DTS WHERE MCAL_CAL_ID IS NULL or MCAL_CAL_ID !=
'Retail Calendar~41'

-- Checking duplicate rows, if any. This should not return any rows.
SELECT MCAL_YEAR,MCAL_PERIOD_NAME,count(*) FROM W_MCAL_PERIOD_DTS GROUP BY
MCAL_YEAR,MCAL_PERIOD_NAME having count(MCAL_PERIOD_NAME) > 1


-- Check number of periods per year. Should always be 12.
SELECT MCAL_YEAR,count(MCAL_PERIOD_NAME) FROM W_MCAL_PERIOD_DTS GROUP BY
MCAL_YEAR ORDER BY MCAL_YEAR
SELECT MCAL_YEAR,count(MCAL_PERIOD) FROM W_MCAL_PERIOD_DTS GROUP BY MCAL_YEAR
ORDER BY MCAL_YEAR

-- After Load procedures completed, check following tables
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_MCAL_PERIOD_D
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_MCAL_DAY_D
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_MCAL_WEEK_D

------------------------------------------------
```

```
-- Checks for PRODUCT.csv file load
------------------------------------------------
-- Verify initial product data before staging it further, row counts should
match data file
select * from W_PRODUCT_DTS

-- Check total count, all counts should be same.  This can indirectly check
for nulls in required columns.
SELECT
count(*),count(item),count(distinct(item)),count(item_level),count(tran_level)
,
count(LVL4_PRODCAT_ID),count(LVL4_PRODCAT_UID),count(LVL5_PRODCAT_ID),count(LV
L5_PRODCAT_UID),
count(LVL6_PRODCAT_ID),count(LVL7_PRODCAT_ID),count(LVL8_PRODCAT_ID),count(TOP
_PRODCAT_ID),
count(ITEM_DESC),count(LVL4_PRODCAT_DESC),count(LVL5_PRODCAT_DESC),count(LVL6_
PRODCAT_DESC),
count(LVL7_PRODCAT_DESC),count(LVL8_PRODCAT_DESC),count(TOP_PRODCAT_DESC)
FROM W_PRODUCT_DTS

-- Check individual counts to make sure it aligns with your source data
SELECT
count(*),count(ITEM_PARENT),count(distinct(ITEM_PARENT)),count(ITEM_GRANDPAREN
T),count(distinct(ITEM_GRANDPARENT)) FROM W_PRODUCT_DTS WHERE ITEM_LEVEL = 1
SELECT
count(*),count(ITEM_PARENT),count(distinct(ITEM_PARENT)),count(ITEM_GRANDPAREN
T),count(distinct(ITEM_GRANDPARENT)) FROM W_PRODUCT_DTS WHERE ITEM_LEVEL = 2
SELECT
count(*),count(ITEM_PARENT),count(distinct(ITEM_PARENT)),count(ITEM_GRANDPAREN
T),count(distinct(ITEM_GRANDPARENT)) FROM W_PRODUCT_DTS WHERE ITEM_LEVEL = 3

-- Checking duplicate rows, if any. This should not return any rows.
SELECT item,count(1) FROM W_PRODUCT_DTS GROUP BY item having count(1) > 1

-- Check item_level, should not have NULL, should have values only 1,2 or 3.
Make sure Count makes sense
SELECT item_level, count(*) FROM W_PRODUCT_DTS GROUP BY item_level ORDER BY 1

-- Check tran_level, should not have NULL, should have only one value for our
purpose. Make sure Count makes sense
SELECT tran_level, count(*) FROM W_PRODUCT_DTS GROUP BY tran_level ORDER BY 1

-- After DTS to DS job executed, check following tables for data
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_PRODUCT_DS
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_PRODUCT_DS_TL
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_PROD_CAT_DHS
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_DOMAIN_MEMBER_DS_TL

-- Expect records for "MCAT" which is the product hierarchy labels code
SELECT DOMAIN_CODE, DOMAIN_TYPE_CODE,LANGUAGE_CODE,
SRC_LANGUAGE_CODE,count(1)
FROM W_DOMAIN_MEMBER_DS_TL GROUP BY DOMAIN_CODE,
```

```
DOMAIN_TYPE_CODE,LANGUAGE_CODE, SRC_LANGUAGE_CODE

-- After Load procedures completed, check following tables
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from w_prod_cat_dh
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from w_product_d
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from w_product_d_tl

-- check for MCAT records for hierachy labels, should align with hierachy
level counts
select domain_code,count(*) from W_DOMAIN_MEMBER_LKP_TL group by domain_code


------------------------------------------------
-- Checks for ORGANIZATION.csv file load
------------------------------------------------
-- Verify initial location data before staging it further, row counts should
match data file
SELECT * FROM W_INT_ORG_DTS

-- Check total count, all counts should be same.  This can indirectly check
for nulls in required columns.
SELECT
count(*),count(ORG_NUM),count(distinct(ORG_NUM)),count(ORG_TYPE_CODE),count(CU
RR_CODE),
count(ORG_HIER10_NUM),count(ORG_HIER11_NUM),count(ORG_HIER12_NUM),count(ORG_HI
ER13_NUM),
count(ORG_TOP_NUM),count(ORG_DESC),count(ORG_HIER10_DESC),count(ORG_HIER11_DES
C),
count(ORG_HIER12_DESC),count(ORG_HIER13_DESC),count(ORG_TOP_DESC) FROM
W_INT_ORG_DTS

-- Checking duplicate rows, if any. This should not return any rows.
SELECT ORG_NUM,count(1) FROM W_INT_ORG_DTS GROUP BY ORG_NUM having count(1) >
1

-- Check ORG_TYPE_CODE, CURR_CODE should not have nulls
-- ORG_TYPE_CODE should be either "S" for Store or "W" for Warehouse
SELECT ORG_TYPE_CODE, CURR_CODE, count(*) FROM W_INT_ORG_DTS GROUP BY
ORG_TYPE_CODE, CURR_CODE ORDER BY 2,1

-- After DTS to DS job executed, check following tables for expected data
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_INT_ORG_DS
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_INT_ORG_DS_TL
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_INT_ORG_DHS
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) FROM W_DOMAIN_MEMBER_DS_TL
-- Expect records for "RTL_ORG" which is the location hierarchy labels code
SELECT DOMAIN_CODE, DOMAIN_TYPE_CODE,LANGUAGE_CODE,
SRC_LANGUAGE_CODE,count(1)
FROM W_DOMAIN_MEMBER_DS_TL GROUP BY DOMAIN_CODE,
```

```
DOMAIN_TYPE_CODE,LANGUAGE_CODE, SRC_LANGUAGE_CODE

-- Org hierarchy level validation on DHS table before loading it to DH
-- On first loading a new hierarchy, this must return with zero issues in the
level structure.  DO NOT proceed if any issues show up.
SELECT '1',
                 'LOCATION org_hier9_num' LEVEL_DESC,
                 location_a.org_hier9_num C_LEVEL,
                 location_a.org_hier10_num P1_LEVEL,
                 location_a.org_hier11_num P2_LEVEL,
                 location_a.org_hier12_num P3_LEVEL,
                 location_a.org_hier13_num P4_LEVEL,
                 location_a.org_top_num P5_LEVEL
            FROM w_int_org_dhs location_a,  w_int_org_dhs location_b
           WHERE location_a.level_name = 'LOCATION'
             AND location_b.level_name = location_a.level_name
             AND location_a.org_hier9_num = location_b.org_hier9_num
             AND (location_a.org_hier10_num <> location_b.org_hier10_num
                  or location_a.org_hier11_num <> location_b.org_hier11_num
                  or location_a.org_hier12_num <> location_b.org_hier12_num
                  or location_a.org_hier13_num <> location_b.org_hier13_num
                  or location_a.org_top_num <> location_b.org_top_num)
        UNION ALL
           SELECT '1',
                 'DISTRICT org_hier10_num' LEVEL_DESC,
                 location_a.org_hier10_num C_LEVEL,
                 null P1_LEVEL,
                 location_a.org_hier11_num P2_LEVEL,
                 location_a.org_hier12_num P3_LEVEL,
                 location_a.org_hier13_num P4_LEVEL,
                 location_a.org_top_num P5_LEVEL
            FROM w_int_org_dhs location_a,  w_int_org_dhs location_b
           where location_a.level_name = 'DISTRICT'
             and location_b.level_name = location_a.level_name
             and location_a.org_hier10_num = location_b.org_hier10_num
             and (location_a.org_hier11_num <> location_b.org_hier11_num
                  or location_a.org_hier12_num <> location_b.org_hier12_num
                  or location_a.org_hier13_num <> location_b.org_hier13_num
                  or location_a.org_top_num <> location_b.org_top_num)
        UNION ALL
           SELECT '1',
                 'REGION org_hier11_num' LEVEL_DESC,
                   location_a.org_hier11_num C_LEVEL,
                 NULL P1_LEVEL,
                 NULL P2_LEVEL,
                 location_a.org_hier12_num P3_LEVEL,
                 location_a.org_hier13_num P4_LEVEL,
                 location_a.org_top_num P5_LEVEL
            FROM w_int_org_dhs location_a,  w_int_org_dhs location_b
           where location_a.level_name = 'REGION'
             and location_b.level_name = location_a.level_name
             and location_a.org_hier11_num = location_b.org_hier11_num
             and (location_a.org_hier12_num <> location_b.org_hier12_num
                  or location_a.org_hier13_num <> location_b.org_hier13_num
                  or location_a.org_top_num <> location_b.org_top_num)
        UNION ALL
```

```
            SELECT '1',
                   'AREA org_hier12_num' LEVEL_DESC,
                   location_a.org_hier12_num C_LEVEL,
                   NULL P1_LEVEL,
                   NULL P2_LEVEL,
                   NULL P3_LEVEL,
                   location_a.org_hier13_num P4_LEVEL,
                   location_a.org_top_num P5_LEVEL
              FROM w_int_org_dhs location_a,  w_int_org_dhs location_b
             where location_a.level_name = 'AREA'
               and location_b.level_name = location_a.level_name
               and location_a.org_hier12_num = location_b.org_hier12_num
               and (location_a.org_hier13_num <> location_b.org_hier13_num
                    or location_a.org_top_num <> location_b.org_top_num)
         UNION ALL
            SELECT '1',
                   'CHAIN org_hier13_num' LEVEL_DESC,
                   location_a.org_hier13_num C_LEVEL,
                   NULL P1_LEVEL,
                   NULL P2_LEVEL,
                   NULL P3_LEVEL,
                   NULL P4_LEVEL,
                   location_a.org_top_num P5_LEVEL
              FROM w_int_org_dhs location_a,  w_int_org_dhs location_b
             where location_a.level_name = 'CHAIN'
               and location_b.level_name = location_a.level_name
               and location_a.org_hier13_num = location_b.org_hier13_num
               and location_a.org_top_num <> location_b.org_top_num;

-- After Load procedures completed, check following tables for final
dimension data
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from w_int_org_dh
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from w_int_org_d
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from w_int_org_d_tl

-- check for RTL_ORG records for the descriptions of hierarchy levels
select domain_code,count(*) from W_DOMAIN_MEMBER_LKP_TL group by domain_code


------------------------------------------------
-- Checks on EXCH_RATE.csv file load
------------------------------------------------
select * from w_exch_rate_dts

select * from w_exch_rate_gs

select * from w_exch_rate_g


------------------------------------------------
-- Checks on ATTR.csv and PROD_ATTR.csv file load
------------------------------------------------
select * from w_attr_dts
```

```
select * from w_product_attr_dts
select * from w_rtl_item_grp1_ds
select * from w_rtl_item_grp1_d

------------------------------------------------
-- Check on W_DOMAIN_MEMBER_LKP_TL issues while loading dimensions
------------------------------------------------
--- DOMAIN MEMBER DUPLICATE RECORD ERROR ---
SELECT DOMAIN_CODE,DOMAIN_TYPE_CODE,DOMAIN_MEMBER_CODE,count(1) FROM
W_DOMAIN_MEMBER_DS_TL GROUP BY
DOMAIN_CODE,DOMAIN_TYPE_CODE,DOMAIN_MEMBER_CODE having count(1) > 1


SELECT DOMAIN_TYPE_CODE, DOMAIN_MEMBER_CODE, DOMAIN_MEMBER_NAME FROM
W_DOMAIN_MEMBER_DS_TL WHERE (DOMAIN_CODE,DOMAIN_TYPE_CODE,DOMAIN_MEMBER_CODE)
IN
(SELECT DOMAIN_CODE,DOMAIN_TYPE_CODE,DOMAIN_MEMBER_CODE FROM
W_DOMAIN_MEMBER_DS_TL GROUP BY
DOMAIN_CODE,DOMAIN_TYPE_CODE,DOMAIN_MEMBER_CODE having count(1) > 1)
ORDER BY 1,2,3


------------------------------------------------
-- Checks on SALES.csv file
------------------------------------------------
-- Verify initial sales data before staging it further, check all columns are
populated with expected values (i.e. CTX was properly formed)
select * from W_RTL_SLS_TRX_IT_LC_DY_FTS

-- Should match the record count from last loaded SALES.csv file
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_RTL_SLS_TRX_IT_LC_DY_FTS

-- Should match W_RTL_SLS_TRX_IT_LC_DY_FTS count
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_RTL_SLS_TRX_IT_LC_DY_FS

-- Should match or be close W_RTL_SLS_TRX_IT_LC_DY_FS count
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_RTL_SLS_TRX_IT_LC_DY_F

-- Look for sls_trx_id in stage but not final sales table
select * from
(
select fs.fs_trx_id, tg_trx_id
from
(select /*+ parallel */ sls_trx_id fs_trx_id from w_rtl_sls_trx_it_lc_dy_fs)
fs left outer join
(select sls_trx_id tg_trx_id from w_rtl_sls_trx_it_lc_dy_f union all select
sls_trx_id from e$_w_rtl_sls_trx_it_lc_dy_tmp) tg
on (fs.fs_trx_id = tg.tg_trx_id)
) where tg_trx_id is null;

-- Look for sls_trx_id/prod_it_wid in stage but not final sales table
select * from
(
```

```
select fs.prod_it_wid, fs.fs_trx_id,  prod_wid, tg_trx_id
from
(select /*+ parallel */ prod_it_wid, sls_trx_id fs_trx_id from
w_rtl_sls_trx_it_lc_dy_tmp) fs left outer join
(select prod_wid, sls_trx_id tg_trx_id from w_rtl_sls_trx_it_lc_dy_f) tg
on (fs.prod_it_wid = tg.prod_wid and fs.fs_trx_id = tg.tg_trx_id)
) where tg_trx_id is null or prod_wid is null;


------------------------------------------------
-- Checks on INVENTORY.csv file
------------------------------------------------
-- Verify initial inventory data before staging it further, check all columns
are populated with expected values (i.e. CTX was properly formed)
select * from W_RTL_INV_IT_LC_DY_FTS

-- Should match the record count from last loaded INVENTORY.csv file
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_RTL_INV_IT_LC_DY_FTS

-- Check that data is making it to target tables after load
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_RTL_INV_IT_LC_DY_F
select /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
count(*) from W_RTL_INV_IT_LC_WK_A

-- inventory validation check after rejected records occur (Support needs to
run this currently, APEX doesn't provide E$ or TMP tables)
SELECT 'E$_W_RTL_INV_IT_LC_DY_TMP','PROD_IT_NUM' DIMM_NAME,PROD_IT_NUM
DIMM_VALUE ,NULL,CHECK_DATE,EFFECTIVE_FROM_DT,EFFECTIVE_TO_DT,'INVALID_PROD_IT
_NUM' INVALID_REASON,NULL,NULL
FROM
(SELECT DISTINCT PROD_IT_NUM,TRUNC(CHECK_DATE) CHECK_DATE, NULL
EFFECTIVE_FROM_DT, NULL EFFECTIVE_TO_DT FROM E$_W_RTL_INV_IT_LC_DY_TMP WHERE
PROD_IT_NUM NOT IN (SELECT PROD_IT_NUM FROM W_PRODUCT_D_RTL_TMP) UNION ALL
SELECT DISTINCT DIMM.PROD_IT_NUM, TRUNC(ERR.CHECK_DATE) CHECK_DATE,
DIMM.SRC_EFF_FROM_DT EFFECTIVE_FROM_DT, DIMM.SRC_EFF_TO_DT EFFECTIVE_TO_DT
from W_PRODUCT_D_RTL_TMP DIMM, E$_W_RTL_INV_IT_LC_DY_TMP ERR WHERE
DIMM.PROD_IT_NUM = ERR.PROD_IT_NUM AND (ERR.DAY_DT > DIMM.SRC_EFF_TO_DT OR
ERR.DAY_DT < DIMM.SRC_EFF_FROM_DT) AND NOT EXISTS (SELECT 1 FROM
W_PRODUCT_D_RTL_TMP DIMM1 WHERE ERR.PROD_IT_NUM = DIMM1.PROD_IT_NUM AND
(ERR.DAY_DT <= DIMM1.SRC_EFF_TO_DT AND ERR.DAY_DT >= DIMM.SRC_EFF_FROM_DT)))
UNION ALL
SELECT 'E$_W_RTL_INV_IT_LC_DY_TMP','ORG_NUM' DIMM_NAME,ORG_NUM
DIMM_VALUE ,NULL,CHECK_DATE,EFFECTIVE_FROM_DT,EFFECTIVE_TO_DT,'INVALID_ORG_DH_
NUM' INVALID_REASON,NULL,NULL
FROM
(SELECT DISTINCT ORG_NUM,TRUNC(CHECK_DATE) CHECK_DATE, NULL
EFFECTIVE_FROM_DT, NULL EFFECTIVE_TO_DT FROM E$_W_RTL_INV_IT_LC_DY_TMP WHERE
ORG_NUM NOT IN (SELECT ORG_NUM FROM W_INT_ORG_DH_RTL_TMP) UNION ALL SELECT
DISTINCT DIMM.ORG_NUM, TRUNC(ERR.CHECK_DATE) CHECK_DATE,
DIMM.EFFECTIVE_FROM_DT EFFECTIVE_FROM_DT, DIMM.EFFECTIVE_TO_DT
EFFECTIVE_TO_DT from W_INT_ORG_DH_RTL_TMP DIMM, E$_W_RTL_INV_IT_LC_DY_TMP ERR
WHERE DIMM.ORG_NUM = ERR.ORG_NUM AND (ERR.DAY_DT > DIMM.EFFECTIVE_TO_DT OR
ERR.DAY_DT < DIMM.EFFECTIVE_FROM_DT) AND NOT EXISTS (SELECT 1 FROM
W_INT_ORG_DH_RTL_TMP DIMM1 WHERE ERR.ORG_NUM = DIMM1.ORG_NUM AND (ERR.DAY_DT
```

```
<= DIMM1.EFFECTIVE_TO_DT AND ERR.DAY_DT >= DIMM.EFFECTIVE_FROM_DT)))
UNION ALL
SELECT 'E$_W_RTL_INV_IT_LC_DY_TMP','ORG_NUM' DIMM_NAME,ORG_NUM
DIMM_VALUE,NULL,CHECK_DATE,EFFECTIVE_FROM_DT,EFFECTIVE_TO_DT,'INVALID_ORG_NUM'
 INVALID_REASON,NULL,NULL
FROM
(SELECT DISTINCT ORG_NUM,TRUNC(CHECK_DATE) CHECK_DATE, NULL
EFFECTIVE_FROM_DT, NULL EFFECTIVE_TO_DT FROM E$_W_RTL_INV_IT_LC_DY_TMP WHERE
ORG_NUM NOT IN (SELECT ORG_NUM FROM W_INT_ORG_D_RTL_TMP) UNION ALL SELECT
DISTINCT DIMM.ORG_NUM, TRUNC(ERR.CHECK_DATE) CHECK_DATE,
DIMM.EFFECTIVE_FROM_DT EFFECTIVE_FROM_DT, DIMM.EFFECTIVE_TO_DT
EFFECTIVE_TO_DT from W_INT_ORG_D_RTL_TMP DIMM, E$_W_RTL_INV_IT_LC_DY_TMP ERR
WHERE DIMM.ORG_NUM = ERR.ORG_NUM AND (ERR.DAY_DT > DIMM.EFFECTIVE_TO_DT OR
ERR.DAY_DT < DIMM.EFFECTIVE_FROM_DT))
UNION ALL
SELECT 'E$_W_RTL_INV_IT_LC_DY_TMP','DAY_DT'
DIMM_NAME,TO_CHAR(DAY_DT,'YYYYMMDD') DIMM_VALUE,
NULL,CHECK_DATE,EFFECTIVE_FROM_DT,EFFECTIVE_TO_DT,'INVALID_DAY_DT'
INVALID_REASON,NULL,NULL
FROM
(SELECT DISTINCT DAY_DT,TRUNC(CHECK_DATE) CHECK_DATE, NULL EFFECTIVE_FROM_DT,
NULL EFFECTIVE_TO_DT FROM E$_W_RTL_INV_IT_LC_DY_TMP WHERE (DATASOURCE_NUM_ID,
DAY_DT)  NOT IN (SELECT MDAY.DATASOURCE_NUM_ID,MCAL_DAY_DT FROM W_MCAL_DAY_D
MDAY,W_MCAL_CONTEXT_G MTEXT WHERE MDAY.MCAL_CAL_WID=MTEXT.MCAL_CAL_WID AND
MTEXT.ORG_ID IN (Select PARAM_VALUE From C_ODI_PARAM Where PARAM_NAME =
'ORG_ID' AND SCENARIO_NAME = 'GLOBAL') AND MTEXT.CALENDAR_ID IN (Select
PARAM_VALUE From C_ODI_PARAM Where PARAM_NAME = 'CALENDAR_ID' AND
SCENARIO_NAME = 'GLOBAL')));
```

F
Accessibility

This section documents support for accessibility in the Retail Analytics and Planning solutions. It describes the support for accessibility and assistive technologies within the underlying technology used by the solutions. Additionally, it covers any accessibility support and considerations built into the application beyond the capabilities of the underlying platform.

ADF-Based Applications

The central user interface for the AI Foundation Cloud Services is built using ADF Faces. Application Development Framework (ADF) Faces user-interface components have built-in accessibility support for visually and physically impaired users. User agents such as a web browser rendering to nonvisual media such as a screen reader can read component text descriptions to provide useful information to impaired users.

ADF Faces provides two levels of application accessibility support:

- **Default:** By default, ADF Faces generates components that have rich user interface interaction, and are also accessible through the keyboard.

  > **Note:**
  >
  > In the default mode, screen readers cannot access all ADF Faces components. If a visually impaired user is using a screen reader, it is recommended to use the **Screen Reader** mode

- **Screen Reader:** ADF Faces generates components that are optimized for use with screen readers. The Screen Reader mode facilitates the display for visually impaired users, but will degrade the display for sighted users (without visual impairment).

Additional fine-grained accessibility levels as described below are also supported:

- **High-contrast:** ADF Faces can generate high-contrast–friendly visual content. High-contrast mode is intended to make ADF Faces applications compatible with operating systems or browsers that have high-contrast features enabled. For example, ADF Faces changes its use of background images and background colors in high-contrast mode to prevent the loss of visual information.

  > **Note:**
  >
  > ADF Faces' high-contrast mode is more beneficial if used in conjunction with your browser's or operating system's high-contrast mode. Also, some users might find it beneficial to use large-font mode along with high-contrast mode.

- **Large-fonts:** ADF Faces can generate browser-zoom-friendly content. In default mode, most text and many containers have a fixed font size to provide a consistent and defined look. In large-font mode, text and containers have a scalable font size. This allows ADF Faces both to be compatible with browsers that are set to larger font sizes and to work with browser-zoom capabilities.

> **Note:**
>
> If you are not using large-font mode or browser-zoom capabilities, you should disable large-font mode. Also, some users might find it beneficial to use high-contrast mode along with the large-font mode.

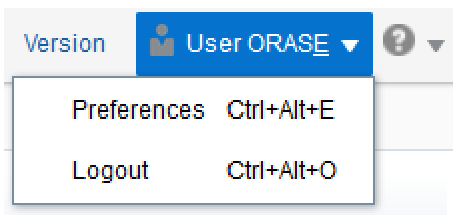
AIF provides the ability to switch between the above accessibility support levels in the application, so that users can choose their desired type of accessibility support, if required. It exposes a user preferences screen in which the user can specify the accessibility preferences/ mode which will allow the user to operate in that mode.

Configuring Application for Screen Reader Mode

Users can configure their session to the accessibility mode by setting user references on the home page of the application as shown below. Perform the following procedure to configure a user preference for screen reader mode.

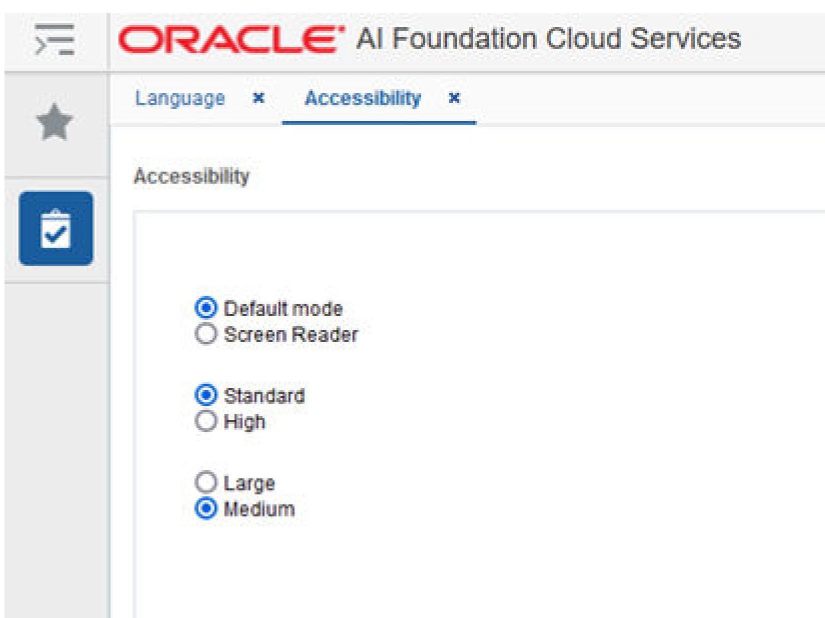
1. Log into the AIF application. Close any open tabs, as they will prevent the accessibility screen from opening.
2. Select **Preferences** from the logged-in user menu in the application home page.

Figure F-1 Logged-in User Menu



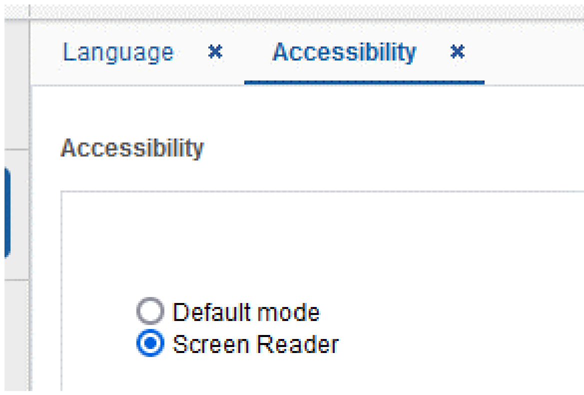
3. Click **Accessibility** in the Tasks pane to open the Accessibility tab.

Figure F-2 Accessibility Tab



4. Select **Screen Reader** to enable accessibility mode, and click **Save**.

Figure F-3 Enabling the Screen Reader



5. Click **Back to Home** to return to the home page.

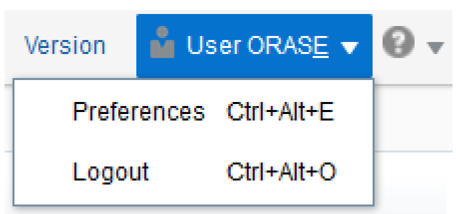
   Now the application is enabled in the screen reader mode to assist a vision-challenged user. Some of the graphical content is also displayed in a tabular mode.

Setting Accessibility to Default

Perform the following procedure to set Accessibility mode to Default mode.

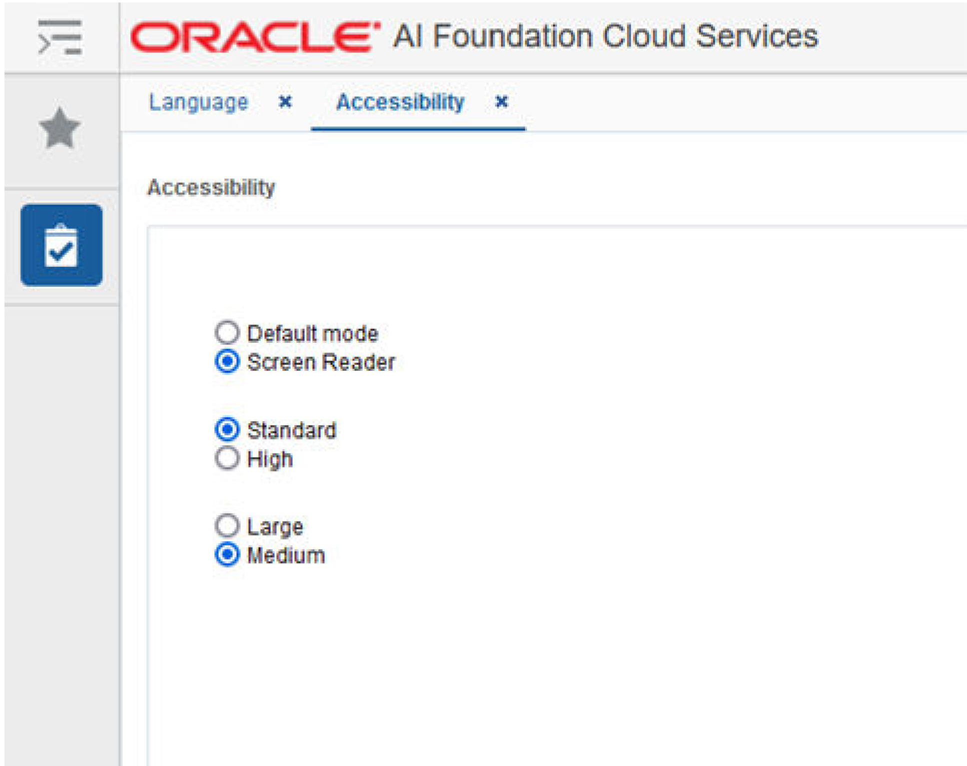
1. From the application home page, select **Preferences** from the logged in user menu.

Figure F-4 Logged-in User Menu



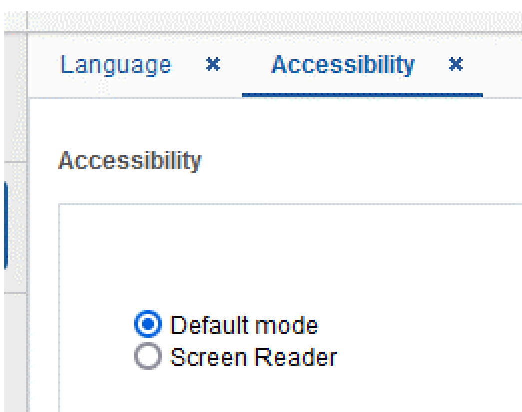
2. Click **Accessibility** in the Tasks pane to open the Accessibility tab.

Figure F-5 Accessibility Tab



3. Select **Default mode**.

Figure F-6 Accessibility Settings



4. Click **Save** to save the settings.
5. Follow the same sequence of steps for the High Contrast and Font Size options within the same screen, as needed.

JET-Based Applications

Some components of the AI Foundation solutions (such as Profile Science and Inventory Planning Optimization) and the interface for the Planning solutions are built using Oracle JavaScript Extension Toolkit (JET).

Oracle JET components have built-in accessibility support that conforms to the Web Content Accessibility Guidelines version 2.0 at the AA level (WCAG 2.0 AA), developed by the World Wide Web Consortium (W3C).

> **Note:**
>
> Because different browsers themselves support accessibility somewhat differently, user experience tends to differ on different web-browsers.

Oracle JET components provide support for:

- Keyboard and touch navigation

  Oracle JET components follow the Web Accessibility Initiative - Accessible Rich Internet Application (WAI-ARIA) guidelines.
- Zoom

  Oracle JET supports browser zooming up to 200%.
- Screen reader

  Oracle JET supports screen readers such as JAWS, Apple VoiceOver, and Google Talkbalk by generating content that complies with WAI-ARIA standards, and no special mode is needed.
- Oracle JET component roles and names

  Each Oracle JET component has an appropriate role, such as button, link, and so on, and each component supports an associated name (label), if applicable.
- Sufficient color contrast

  Oracle JET provides the Alta theme which is designed to provide a luminosity contrast ratio of at least 4.5:1.

OAS-Based Applications

Retail Insights uses the Oracle Analytics Server as its user interface, and benefits from all the native accessibility features added to that platform. For details on the accessibility features in OAS, refer to the Accessibility Features and Tips chapter in the *Oracle® Analytics Visualizing Data in Oracle Analytics Server* guide.

RPASCE Configuration Tools

The configuration tools for the Retail Predictive Application Server Cloud Edition (RPASCE) is a separate component used with the Planning applications and it has its own set of accessibility features. Refer to the Accessibility chapter in the *Oracle Retail Predictive Application Server Cloud Edition Configuration Tools User Guide*.

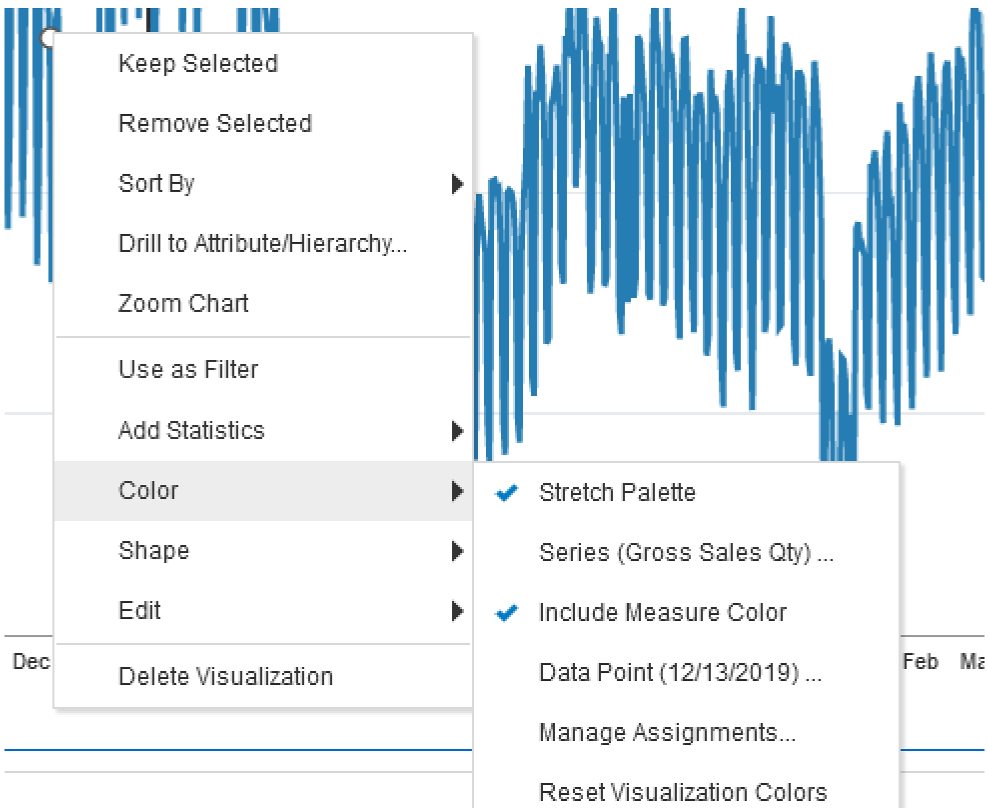
Report Authoring Guidelines

Users of the Retail Analytics and Planning solutions may leverage Oracle Analytics for creating custom content such as reports and dashboards. It is possible to develop this content with accessibility in mind without sacrificing any features or functionality. Some general guidelines for creating accessible content are provided below.

Color Usage in Tables and Graphs

The default set of color pallets used in Oracle Analytics for cell shading and visualizations are designed with accessibility in mind and should provide sufficient contrast between on-screen elements. If you are choosing your own colors, avoid using multiple similar colors for data elements, and do not convey important information solely through the color of the element. To select a color range or specify your own, right-click on a visual element and use the Color menu option for Manage Assignments.

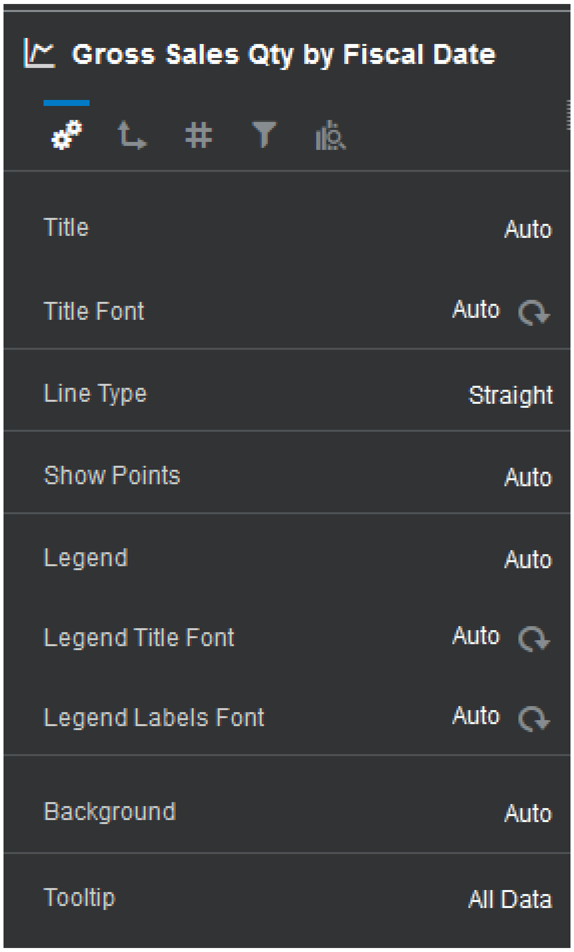
Figure F-7   Color Properties



Text and Label Usage

By default, every table or graph view added to a report will have a title included. Use the text properties of each view to add additional text such as graph labels, hover text, and custom view headers. All views should include these textual elements to better support users who are visually impaired or leveraging an accessibility feature like large fonts or screen readers. To access the text properties, click the view and navigate to the left-side panel that appears.

Figure F-8 View Properties



Layout and Canvas Usage

Oracle Analytics allows you to arrange multiple views within a single on-screen layout, as well as create multiple canvases with tabs and longer report layouts with automatic scrollbars added as needed. It is best to keep each canvas simple and focused on only a couple of views. Avoid using a large number of small charts or tables placed tightly together on-screen and avoid requiring the user to scroll down the page repeatedly to see all the visuals. Create multiple canvases to simplify your layouts and use a similar arrangement of data in each canvas to make it easy to navigate and read.

Figure F-9 Example of Multiple Canvas Tabs