Oracle® Retail Markdown Optimization

Implementation Guide 1

Release 13.1

July 2009

ORACLE®

Oracle® Retail Markdown Optimization Implementation Guide, Release 13.1

Primary Author: Judith Meskill

Contributing Author:

Contributor:

(vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

4 MDO Data

Preface

The Oracle Retail Markdown Optimization Implementation Guide provides instructions on how to implement the Markdown Optimization Product.

Audience

This document is intended for implementers of Markdown Optimization.

Related Documents

For more information, see the following documents in the Oracle Retail Markdown Optimization documentation set

- *Oracle Retail Markdown Optimization Installation Guide*
- *Oracle Retail Markdown Optimization User Guide*
- *Oracle Retail Markdown Optimization Administration Guide*
- *Oracle Retail Markdown Optimization Configuration Guide*
- *Oracle Retail Markdown Optimization Operations Guide*
- *Oracle Retail Markdown Optimization Release Notes*

or in the Oracle Retail Analytic Parameter Calculator Markdown Optimization documentation set:

- *Oracle Retail Analytic Parameter Calculator Markdown Optimization Installation Guide*
- *Oracle Retail Analytic Parameter Calculator Markdown Optimization User Guide*
- *Oracle Retail Analytic Parameter Calculator Markdown Optimization Configuration Guide*
- *Oracle Retail Analytic Parameter Calculator Markdown Optimization Release Notes*

Customer Support

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

- https://metalink.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 recreate
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

If you are installing the application for the first time, you install either a base release (for example, 13.0) or a later patch release (for example, 13.0.2). If you are installing a software version other than the base release, be sure to read the documentation for each patch release (since the base release) before you begin installation. Patch documentation can contain critical information related to the base release and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site (with the exception of the Data Model which is only available with the release packaged code):

http://www.oracle.com/technology/documentation/oracle_retail.html

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Getting Started

This chapter introduces the Oracle Retail Markdown Optimization (MDO) *Implementation Guide*. The Implementation Guide provides an overview of the tasks involved in configuring and implementing MDO.

Introduction

The MDO *Implementation Guide* is intended for use with the rest of the MDO documentation set and so does not duplicate the information found in those documents.

It should be used in conjunction with the following documentation:

- *Oracle Retail Markdown Optimization Installation Guide*
- *Oracle Retail Markdown Optimization User Guide*
- *Oracle Retail Markdown Optimization Administration Guide*
- *Oracle Retail Markdown Optimization Configuration Guide*
- *Oracle Retail Markdown Optimization Operations Guide*
- *Oracle Retail Markdown Optimization Release Notes*

The MDO Installation Guide provides detailed instructions about planning the installation, setting up the database, setting up the application server, installing the MDO application, setting up the BI structure, and setting up single sign on.

The MDO User Guide provides step-by-step procedures for all UI-related tasks.

The MDO Administration Guide provides information about managing users (User Management), managing business rules (Business rule manager), and using the Seasonality Manager.

The MDO Configuration Guide is a technical reference guide that provides technical details about all MDO configuration points.

The MDO Operations Guide is a technical reference guide that provides technical details about the standard interface, the standard load process, the model run, MDO tools, and troubleshooting.

The MDO Release Notes provides a high-level overview of new features and significant issues for the latest release of the MDO application.

MDO Analytical Parameter Calculator

MDO comes bundled with Analytic Parameter Calculator Markdown Optimization (APC MDO).

This application is a single-user application that calculates the demand parameters required to perform forecasting. It uses the historical data to calculate and derive the demand parameters.

APC MDO generates a set of text files that contain the calculated demand parameters. These parameters are loaded into the MDO schema during implementation of MDO.

The MDO Implementation Guide provides a high-level overview of the technical concepts and processes involved in calculating analytical parameters for MDO. However, the complete analytical process is beyond the scope of this document.

The APC MDO documentation set consists of the following:

- *Oracle Retail Analytic Parameter Calculator Markdown Optimization Installation Guide*
- *Oracle Retail Analytic Parameter Calculator Markdown Optimization User Guide*
- *Oracle Retail Analytic Parameter Calculator Markdown Optimization Configuration Guide*
- *Oracle Retail Analytic Parameter Calculator Markdown Optimization Release Notes*

The APC MDO Installation Guide provides detailed instructions about planning the installation, setting up the database, setting up the application server, installing the application on OAS, and installing the application of WebLogic.

The APC MDO User Guide provides step-by-step procedures for all UI-related tasks.

The APC MDO Configuration Guide is a technical reference guide that provides technical details about data requirements, user management, and other configuration topics.

APC MDO uses the same standard interface and standard load procedures as MDO. Refer to the MDO documentation for information about these topics.

The APC MDO Release Notes provides a high-level overview of new features and significant issues for the latest release of the APC MDO software.

How This Guide is Organized

The MDO *Implementation Guide* is organized into the following chapters.

1. Introduction
2. Implementation Tasks – a discussion of business requirements and a list of tasks that must be completed during an implementation.
3. Analytics – an introduction to analytical concepts and an overview of the MDO APC tool.
4. MDO Data – a summary of the standard interface, the standard load, and data transfer guidelines.
5. Business Rules – a discussion of the details regarding the configuration of specific business rules.
6. Inference Rules – a discussion of the details regarding the configuration of significant inference rules and some examples.
7. Front End Configuration – the basics of configuring the UI.
8. Model Run – a discussion of the configuration details pertaining to the model run.
9. User Management – a discussion of user roles and password policies.
10. Localization – overview of key localization concepts.

11. Production – a high-level discussion of the production process, automation, and the weekly production schedule.
12. Best Practices – implementation recommendations.

2

Implementation Tasks

This chapter contains:

About the MDO Implementation

The focus of the MDO application is the weekly model run batch process. This process uses the forecasting model produced by APC MDO to perform an optimization using the business rules. The optimization calculations produce recommendations regarding when and how deeply to discount merchandise in order to achieve the highest gross margin dollars over the entire life cycle to the defined out date. Key performance indicators, which are metrics such as Gross Margin, are also produced. These metrics are used to populate the MDO UI display.

The implementation process creates a client-specific configuration. The configuration is determined by as client's individual business requirements.

The MDO Calc Engine evaluates every possible price trajectory in order to find the best price that satisfies all the business rules.

The MDO forecasting model takes into account seasonality, price elasticity, promotions, and inventory effects. Historical and in-season data are combined to produce the forecasting model.

The Calc Engine tries thousands of markdown scenarios with resulting sales and forecasts as it searches for the optimal price cut within the business rule constraints.

Markdowns have the greatest impact early in the season when client demand for the merchandise is the greatest.

This chapter provides information about the implementation process. It includes a list of all the necessary tasks as well as useful details to support the process.

Business Requirements

The configuration of MDO is determined by a client's individual business requirements. A complete and accurate configuration is essential to the accuracy and success of MDO markdown recommendations and forecasts.

These business requirements include:

- The client's pricing and markdown strategy.
- The following information regarding the client's data:
  - The level of the merchandise hierarchy and of the location hierarchy at which the sales data will be provided.
  - The level of the merchandise and location hierarchies at which the item is defined.
  - The level at which the worksheet should be managed.
  - The number of regions or stores that exist at the item level (the level of optimization).
  - Whether markdowns will be taken differently by color.
  - Whether markdowns will be taken differently by cluster.
  - The day of the week that markdowns are effective.
  - Whether markdowns are effective on the same day for all departments.
  - Whether promotions, budget, and distribution center data will be included in the data feed.
  - Whether markdowns taken data will be provided.
  - How historical markdowns will be provided.
  - The number of sales records that will be provided each week.
  - The number of active records that exist at the item level.
  - How eligible items are defined (for example, by time since exit date or threshold number of units on hand).
  - The kinds of inventory that should be considered by the batch process for optimization.
  - The number of sendback files that are required and in what format they should be provided.

Hierarchy Levels

The client's retail merchandise and retail location structure must be described. (See the MDO *Operations Guide* for operational details.)

For example, the merchandise hierarchy may consist of:

1. Chain
2. Division
3. Department
4. Class
5. Style
6. Color

And the location hierarchy may consist of:

1. Chain
2. Region
3. District

4. Store

Optimization Level

The optimization level (termed the item) is the intersection of the merchandise and location nodes at which the optimizations will be calculated. This is defined once at the start of the configuration using the Cross Product Information and MH/LHLevels standard interface. See the Standard Load chapter in the MDO *Operations Guide* for details.

The level used in each hierarchy is a configuration choice, but must be the same for all items. For example, items might be defined at the Color-Region level. MDO aggregates sales data to the item level and persists sales data at the item level only.

Data Feeds

It is essential that the data provided by the client be complete and formatted correctly. Three years of historical data are required for the initial determination of the analytical parameters and the implementation. In the production environment, merchandise hierarchy, location hierarchy, sales data, and must be provided as part of the weekly data feed.

Weekly Schedule

The MDO process is a weekly one. Typically, the model run batch process is scheduled to occur during the weekend. At this time, the application is not available to users. During the week, the application is not available at certain times, such as when the database is being backed up. In addition, certain user activities, such as submitting markdowns for approval, must be completed according to a defined schedule. Each client's schedule can vary and may be constrained if the client is operating in a hosted environment.

Metrics

Metrics are calculations that support the optimization process. These metrics are displayed within the UI in the Worksheets. Each client will use a combination of standard metrics and custom metrics. The custom metrics must be identified and defined.

Project Roles

Some of the possible project roles in an MDO implementation include the solution architect, technical deployment manager, the business consultant, the analytics project manager, the technical lead, the database engineer, and the project manager.

The client is responsible for providing an executive manager to represent the client as well as a project manager, a business lead, a database administrator, and a technical lead.

Business Constraints

The following client-specific information should be collected as part of requirements gathering.

- Model Start Date. How the model start date is defined is different for each client. Typically, it is set to the date when the sales data becomes meaningful.See IR_MODEL_START_OPTION.

- Exit Date. MDO requires an exit date in order to provide recommendations. Basic items do not have an exit date. Target inventory levels and salvage values must be associated with the exit date or MDO assumes a value of 0. The recommendations that MDO makes are aligned with a specific exit date. MDO does not provide any recommendations past the item's exit date.
- Markdown Cadence. The inference rules are used to configure markdown characteristics such as the calendar day of the week when markdowns are effective and whether or not markdowns are effective on the same day for all departments.
- Blackout Period. MDO removes the dates during which markdowns are prohibited (for example, during holidays) from the eligible markdown calendar. If MDO determines that a markdown is warranted during a blackout period, it will make a recommendation either before or after that period.
- Business Rules. see the chapter on Business Rules for information about this configuration.
- Pricing Groups. Used to optimize similar items at the same time. Questions revolve around which items will be grouped and how. Typically, items in a pricing group share a common attribute such as vendor. Items within an pricing group must be priced at the same price point of the same % off at the same time. MDO uses pricing group in the determination of the optimal pricing strategy for the entire group. The optimal pricing strategy for the group may be sub-optimal for items within the group. MDO manages pricing groups at the item level. However, pricing groups can be managed at the CHAIN level and optimized at the item level. CHAIN level pricing groups can be useful when adding merchandise to a pricing group at all locations.
- Flexible Store Clustering is an optional feature of Markdown Optimization that permits clients to group stores differently for different sets of merchandise. Grouping the stores in this way can facilitate more accurate optimizations and forecasts than can occur at the chain level, because the selling patterns for the set of merchandise in the stores of a given cluster will be similar. Analytical Services is responsible for the design of a client's flexible store clustering configuration. Flexible Store Clustering is implemented in MDO via the Standard Interface and the Standard Load.

Price Ladders

Price ladders are used to specify what price amounts and what price discounts a client will use and are what MDO uses for markdowns. This information is required in order for MDO to generate forecasts and recommend markdowns that are consistent with the client's business constraints. Price ladders are configured using IR_PRICE_ LADDER. Prices are defined relative to the original full retail price.

The price ladder types are:

- Permanent Price Point – use permanent accounting; specify a dollar amount.
- Permanent Percent Off – use permanent accounting; specify a percent of the full price
- Temporary Price Point – use temporary accounting; specify a dollar amount
- Temporary Percent Off – use temporary accounting; specify as a percent of the full price
- POS percent – use temporary accounting; specify as a percent off the current ticket price

All recommended markdown prices must exist on the Price Ladder. Each item will be assigned a list of valid item price points. MDO's markdown recommendations must correspond to one of these price points. Prices will be configured accordingly. A price ladder is a set of values that prescribe all possible prices for a particular client's merchandise (items and collections).

A price point ladder has a set of fixed prices, such as the following:

- 1.99
- 2.99
- 3.99
- 5.99
- 9.99

The following is an example of a price-percentage ladder, which specifies percentages that are allowed as definitions for price markdowns:

- 75%
- 67%
- 50%
- 33%
- 20%
- 10%

If the Calc Engine generates a recommended price that is not on the client's price ladder, MDO converts the amount to the next most expensive price.

Implementation Project Plan

The responsibility column and the status column have purposely been left blank so that you can use this form as a template. The steps are generally ordered, but can be used in a flexible manner in response to a specific client profile.

Table 2–1 Implementation Tasks

Task	Person Responsible	Status
Planning		
Contract signed.		
Assign resources.		
Develop preliminary project dates.		
Kickoff meeting.		
Network planning.		
Hardware planning.		
Data planning		
Business Requirements		
Define optimization level.		
Define business rules.		
Define screens and views.		

Table 2–1 (Cont.) Implementation Tasks

Task	Person Responsible	Status
Define price ladders.		
Define outdate logic.		
Define weekly operational schedule.		
Define metrics.		
Define pricing groups.		
Define flexible clustering.		
Define user profiles.		
Promotions defined.		
Define reports.		
Define markdown calendar.		
Review Business Rule Manager functionality.		
Review Seasonality manager functionality.		
Define eligibility.		
Define forecast-able inventory.		
Define model start date.		
Data Requirements		
Review standard interface specification and discuss the requirements for each data feed.		
Review historical data feeds.		
Review weekly data feeds.		
Review data validation.		
Define sendback files.		
Network Requirements		
Review file transfer procedures.		
Review data encryption requirements.		
Conduct test of ftp process.		
Environment and Technical Requirements		
Define production hardware configuration.		
Obtain hardware and software licenses.		
Requirements Documentation		
Develop first draft of requirements document and review with client.		
Define any open issues that arise with requirements document.		
Present final draft of requirements document to client.		
Identify non-standard configuration points.		
Sign-off on requirements documentation.		
Baseline project plan is complete.		

Table 2–1 (Cont.) Implementation Tasks

Task	Person Responsible	Status
Create Development and Test Environment		
Allocate storage.		
Create development environment.		
Create and configure development database.		
Create analytical database (asds).		
Create test environment.		
Create MDO database.		
Import data from development database to test database.		
Set up application environment.		
Install Java.		
Install MDO application.		
Create directory structures and copy files.		
Configure application server.		
Customize build environment.		
Prepare environment for model run.		
Load and validate historical data		
Receive full set of three years of data from the client.		
Load and validate historical data into development environment.		
Load and validate data into analytics environment.		
Calculate first pass of analytical parameters. Refine as necessary.		
Design		
Translate requirements document into design document and review.		
Develop test cases.		
Configuration		
Configure front end screens and grids.		
Configure metrics.		
Configure inference rules.		
Configure load statements.		
Configure data interface.		
Configure reports.		
Configure sendbacks.		
Build		

Table 2–1 (Cont.) Implementation Tasks

Task	Person Responsible	Status
Complete an iterative series of model runs to progressively test configuration points. The following lists possible example; however, the iterative steps will depend on the specific client design.		
Model run 0: Generic data with defined results.		
Model run 1: historical data		
Model run 2: working UI, model run parameters (business rules and inference rules) loaded.		
Model run 3: first analytical configuration		
Model run 4: non-standard configuration, custom metrics.		
Model run 5: weekly data loaded, sendbacks, reports, revised demand parameters.		
Automation		
Develop automation process and scripts.		
Production Environment		
Create production environment.		
Load and validate weekly data.		
Internal Testing		
QA analytical configuration.		
Unit testing.		
Functional testing.		
Model run testing.		
Automation testing.		
Performance testing.		
Certify system ready for UAT.		
Training and UAT		
Test UI configuration.		
Test standard and custom metrics calculations.		
Validate all business rules.		
Complete end-to-end system integration, business processing, and automation testing.		
Validate markdown recommendations and forecasts.		
Validate report configuration.		
completer user training.		
Sign-off.		
Go Live		

Configuration Steps for the Model Run

The configuration of the Model Run is an iterative process in which changes are made in a step-wise fashion beginning with the simplest of the configuration points. After each change, the batch process is run in order to make sure that the changes result in a model run that is both successful and that produces the desired results. If the model run is either not successful or does not produce the expected results, you should modify the configuration and re-run the model run.

- Model Run 0. Conduct Model Run 0 after the application has been installed. Use the default values and the dataset that are provided with the application. The expected results for the model run are provided in the MDO documentation. This model run is used to determine if the application has been successfully installed. Complete the model run and examine the error logs.
- Model Run 1. This model run is used to begin the MDO configuration. It uses a client's historical data for merchandise and hierarchy levels, calendar, items, and sales. Stage and load the client's data according to the Standard Interface and Standard Load information provided in the MDO documentation. Verify that the data has loaded successfully. In addition, configure the following parameters:
  - Populate ASH_CP_TBL in order to define the optimization level.
  - Populate ASH_MHL_TBL in order to set the client's merchandise levels and the order.
  - Populate ASH_LHL_TBL in order to set the client's location levels and the order.
  - Configure IR_WORKSHEET_IDS in order to create the worksheet. This may depend on client requirements. The default values may be sufficient.
  - Do a first pass at configuring item eligibility. IR_ELIGIBLE provides a list of eligible items and pricing groups to the model run. Eligibility is defined and customized via the load statements.
  - Configure other inference rules as necessary. This may also depend on client requirements. The default values may be sufficient.
  - The default model start date is used for this model run. Use dummy values for price ladders, seasonalities, and outdates.

  Complete the model run and examine the error logs.
- Model Run 2. This model run involves the initial configuration of the business rules, certain inference rules, price ladders, and the UI.
  - Use rule_definition.xml to define the initial values for attributes and custom rules (if necessary).
  - Configure IR_BUSINESS_POLICY, which defines business constraints such as markdown depth and salvage values The view looks up most values from the BRM.
  - Configure the following inference rules:
  - Configure price ladders using the Standard Interface and the Standard Load. For custom price ladders, use IR_PRICE_LADDER. For custom price ladders, you must assign custom price ladder logic.
  - Configure the UI basic Worksheet Summary view, the Basic Item Worksheet default views, the correct levels for the price ladders.

Complete the model run and examine the error logs.

- Model Run 3. For this model run,

  Configure the following inference rules:

  - IR_PARAMETERS
  - IR_SEASONALITY_ATTRIBUTE

  For this model run, the analytical parameters must be loaded.

  Load client-specific outdates, demand parameters, and seasonalities.

  Load promotions, configure IR_PLANNED_PROMOS, and configure promotion blackout periods.

  Refine eligibility criteria.

  Complete the model run and examine the error logs.

- Model Run 4. For this model run, configure the following:

  - IR_USER_FLOATS and IR_O_USER_FLOATS.
  - Custom metrics using p4pgui-config.xml, p4p-custom-columns.xml, and p4p-column-list.xml.
  - Configure the UI grids: worksheet summary grid, item worksheet views, add/remove items grid, merchandise maintenance grids (items/groups), promo detail grid, display view, item details layout, whatif view, and item popup view.
  - For pricing groups, IR_ITEM_COLLECTION, IR_COLLECTION_INFO, and ITEM_COLLECTION_OPTION.
  - Error thresholds for the standard load procedure.

  Complete the model run and examine the error logs.

- Model Run 5. Configure reports, custom reports, and sendback files. Complete the model run and examine the error logs.
- Model Run 6. Complete testing and certify configuration.

3

Analytics

This chapter contains the following:

Introduction

The chapter on the analytics provides detailed information about questions clients may ask about how markdown recommendations and forecasts are made. This chapter also provides reference material to supplement the APC MDO documentation.

APC MDO is an analytical, fact-based application designed to be used for retail markdown management. A client's historical sales patterns are analyzed to produce demand models. The resulting demand parameters are used to make optimal markdown recommendations based on specific business goals and retail constraints.

The demand model uses historical sales patterns to capture the following demand parameters:

- Seasonality – defines how sensitive the merchandise is to seasonal influences such as holidays and weather. It also defines the fashion effect.
- Price Effects – defines how sensitive the client is to price changes and how much lift is provided by a given markdown.
- Promotions – defines how different promotions affect store traffic and the sale of full-priced and sale-priced merchandise.
- Inventory Effect – defines how many sales are lost because of stockouts.

The demand parameters are used by the forecasting engine to produce a forecast. APC MDO then uses the Calc Engine and applies the forecast to evaluated millions of scenarios. The output is a set of markdown recommendations.

Based on the forecast, the system models the markdowns that are needed to reach the inventory goals for that item, using the business constraints defined in the Business Rule Manager.

The challenge in managing markdowns is determining when and how deeply to discount the merchandise in order to achieve the highest gross margin by the defined exit date.

The goal is to maximize GM dollars over the entire life cycle, not just during the full price selling period.

Markdowns have the greatest impact early in the season while there is still client demand for the merchandise. Once client demand has diminished, the markdowns are much less effective.

Process Overview

Here is a high-level overview of the analytical process.

1. Collect client's business requirements.
2. Create initial design based on the client profile.
3. Review the client's configuration requirements with the client.
4. Obtain three years of historical sales data from the client.
5. Install the asds data schema and populated it the client data.
6. Validate the data to determine whether or not the data is consistent and reliable.
7. Run APC MDO.
8. Input the output text file from APC MDO into the MDO application in a test environment.
9. Conduct iterative testing of the model run and certify when results are satisfactory.
10. Conduct UAT of the analytics.

Historical Data Load

In order to perform the historical analysis and calibrate the forecast model, three full years of historical sales in a one-time historical feed of hierarchical sales and inventory data are required.

The required data files are described below:

There are five types of data required for the historical data load: Merchandise Hierarchy, Location Hierarchy, Calendar, Items, and Sales / Inventory / Orders. Oracle prefers clients to provide a single Merchandise Hierarchy, Location Hierarchy and Items file for the entire set of historical data, rather than one file for each historical week. All historical sales records should have corresponding entries in the Hierarchy and Items files. If flexible clustering is used, then the Cluster Mapping file is also required.

The standard base names for the files are:

- mh – merchandise hierarchy
- lh – location hierarchy
- cal – fiscal calendar. It is common that a fiscal calendar is provided at least 10 years into the future
- items – item data
- sales – weekly sales and inventory data
- merchclusters – flexible clustering mapping

Oracle recommends receiving the historical sales data broken down by week, producing files with the same names and formats as the weekly sales file. If necessary,

clients can combine multiple historical weeks into monthly or quarterly sales files. A shell script can be written to facilitate loading of the files.

File names should always have a date stamp that corresponds to the year and fiscal week contained in the file. In the event that a sales file contains more than one week of data, the last (most recent) year and week contained in the file should be used as the date stamp.

Loading history data follows the data loading process outlined above. More detailed steps are outlined below:

1. Ensure the setup data loads steps are completed
2. Stage mh, lh, cal, item and cm (if applicable) files
3. Stage the first of sales files
4. Validate the ASH data
5. Load merchandise hierarchy, location hierarchy, calendar, item and cluster mapping data.
6. Load the staged sales data.
7. Resolve loading errors found in the _BAD staging tables
8. Stage and load all subsequent sales files, one at a time

There are two types of data validation: technical data validation and analytical data validation.

Technical data validation ensures that the staging and loading of data is technically sound. During the load process, offending records are logged in a separate table corresponding to the data loaded (e.g. ASH_ITEMS_TBL_BAD when loading items). In addition, it is a good idea to validate data during and after staging. For example, you can check whether:

- File formats and naming conventions are correct
- The interface specification has been adhered to: all required fields are present, keys are unique, and key levels have one and only one parent key. See the Standard Load chapter in the MDO *Operations Guide* for more information.
- Sales data is defined at the correct merchandise and location hierarchy levels
- Item keys are at the correct level of optimization and are present in their corresponding hierarchies

Analytical data validation ensures the quality of data loaded is sufficient to perform analytical parameter calculations. See the MDO APC documentation.

Note that usually both activities entail providing feedback to the retailer regarding the number of items and sales volume in units and value. It is essential that potential problems with the historical data are recognized as early as possible to ensure reliable recommendations.

Using APC MDO

The MDO Analytic Parameter Calculator (MDO APC) is an analytical tool used to calculate demand parameters, seasonality and price elasticity, and view the results. These demand parameters are produced in the format required by MDO.

The APC MDO application is organized into stages. Each stage occupies a separate screen in the UI. Each stage contains parameters that are configurable.

See the APC MDO documentation suite for more information.

The APC MDO Process

The operation of the APC MDO is divided into stages. Each stage gathers related information and calculations together. The results of the computation of each stage can be kept for as long as the computation is valid. The stages can be run in order or a stage can be skipped if the results are still valid. Skipping stages, if a valid option, can improve performance.

Within each stage, you can perform the following three operations:

- Modifying the input values for the stage. Each stage has default values that you can change. Changing default values requires an in-depth understanding of how the APC MDO works and of the retail details of your business. You can modify the fields without immediately running the stage. Note that modifying the Base Historical Period can only occur after the Season Code stage is successfully run. This dependency exists because the Base Period dialog box displays the results of the Season Code stage calculations.
- Running the stage. Each stage performs its calculations during the run operation. Stages cannot be run simultaneously. After you run a stage, you can modify some of the fields in the stage. However, until you re-run the stage, the results of the stage do not change. Thus, you can continue to modify the fields or change them back to original values. It is recommended that you retain the results of a stage for as long as possible.
- Viewing the stage results. You can view the calculated results for the Preprocessing and Pruning stages after that stage has finished running.

You cannot in general run all stages of APC MDO on the very first run. You need at least two runs. In the first run, run the APC MDO through the Smoothing Stage, but not beyond. Based on this first run, set the Base Historic Period (in the Pruning Stage) by visually examining the chain-level seasonality curves using the Raw Seasonality Viewer. If the curve has a sharp fall-off, you should look at the sales dollars. You should make sure you have between 80% and 90% of the original sales dollars. After you have properly set the Base Historic Period, you can run the remaining stages (from Pruning onward). But to set the Base Historic Period from the results of Raw-AP you must run at least two runs when you first use APC MDO with brand-new data. (Of course, after the Base Historic Period is set, you can run all stages in a single run.).

Here is a summary of the complete APC MDO workflow.

1. Logging in – one user can log in at a time. Only one user can use APC MDO at a time.
2. Data Validation – a summary of historical data can be used to check that the historical data has been loaded correctly. This information only needs to be checked when the historical data is new or has been reloaded during a data refresh.
3. Preprocessing – these settings can be changed.
4. Examination of Preprocessing Results – view details about how much data was filtered out by preprocessing. Verify that at least 80% of the sales dollars remain.
5. Season Code Selection – used to define the season code.
6. Levels Selection – within the Raw AP stage, select the merchandise and location levels for which APC MDO will calculate the demand parameters.

7. Entering Raw AP Filtering Parameter – Raw AP filtering (in addition to preprocessing filtering).
8. Running Raw AP Calculation – produces the demand parameters.
9. Examination of Demand Parameters – view seasonality curves using the Raw Seasonality viewer.
10. Performing Smoothing – calculation of seasonality correlations.
11. Entering Base Historical Period – used to override a fiscal year. This can only be set after the Season Code Setup stage is run.
12. Pruning – the pruning of partitions by setting reliability tolerances for demand parameters. This is used to eliminate unreliable partitions.
13. Examination of Pruning Results – view results of the partition filtering by using the Parameter Histogram viewer.
14. Examination of Summary Statistics – click **View Results** to view details about the pruning filtering (the partitions after pruning).
15. Performing Corrections – adjust curves for holidays and promotions.
16. Performing Propagation of Seasonality Curves – copy curves to other fiscal years.
17. Entering Escalation Path – this is used by APC MDOC for the output. The default is to escalation along the merchandise hierarchy first, and then along the location hierarchy.
18. Output Stage – specify the output path.
19. Parameter Export – generate files for other analytic products.

Changing Parameter Settings

The following parameters, labelled with a Caution icon, must be examined before running any of the stages in the APC MDO, in order to determine whether or not the parameter values should be changed. Usually these are the only parameters you will need to change to run APC MDO.

- Season Code Setup
  - Start Date Code
  - End Buckets
  - Map Attributes
- Raw AP
  - Merchandise Hierarchy
  - Location Hierarchy
  - Pruning Base Historical Period
- Corrections
  - Catch-All Curve Fiscal Year
  - Padding Curve Fiscal Year (The fiscal year for this parameter and the above parameter must be a complete fiscal year.)
  - Holidays
  - Promotions (Only enable promotions and holidays that have a large traffic lift.)

- Output – escalation path

Dependencies

Each stage must be run in the order listed in the process train. Each stage is dependent on the previous one (i.e., all previous stages must have a status of complete before the current stage can be run).

4

MDO Data

This chapter contains:

Introduction

MDO provides a standard interface as a set of control files and staging files that define how the data should be sent by the client to MDO. The specifications define all the one-time data and all the weekly data that MDO requires, including analytical parameter data.

MDO provides a standard load procedure to stage and load all that data. The standard load procedure includes detailed validation of the data in the data feeds.

The chapter on the standard interface and standard load provides detailed information about questions clients may ask about configuration as well as high-level information about how to approach configuring MDO data requirements. See the MDO *Operations Guide* for technical configuration details.

Standard Interface

An important part of getting Markdown Optimization up and running in a production environment is the gathering and loading of enterprise data. Markdown Optimization requires historical and weekly data to be loaded into the application database. The data must be provided in a standard format, as specified in the standard interface specification. The data can then be loaded according to the standard load procedure.

This section details the data interface to the application. The application requires that client data be provided in flat files containing pipe-delimited data organized so that the data can be loaded into the application database tables that follow the formats specified here.

The following special characters are not allowed: colon, semi-colon, comma, forward slash, backward slash, any type of quote, any type of apostrophe, <, or >.

Three interfaces (Merchandise Hierarchy Levels, Location Hierarchy Levels, and Cross Product Information) that are required by the application are only loaded once. The information contained in these three files is collected during discussions with specific

clients; however, the files themselves are not provided by clients but are created and loaded as part of the initial Markdown Optimization configuration.

Here is a table of all the standard interfaces

Table 4–1 Interface Specifications

Interface Specification	Required/Optional
Merchandise Hierarchy	Required
Location Hierarchy	Required
Calendar	Required
Items	Required
Sales/Inventory/Orders	Required
Markdowns Taken	Required
Budget	Optional
Promotions	Optional
Distribution Center Inventory	Optional
Distribution Center Allocation	Optional
Merchandise Hierarchy Rename	Optional
Location Hierarchy Rename	Optional
Merchandise Hierarchy CDA	Optional
Location Hierarchy CDA	Optional
Items CDA	Optional
Business Rule Instances	Optional
Demand Parameters	Required
Price Ladders	Required
Seasonalities	Required
Cluster Mapping	Optional

MDO requires a minimum of two full years and preferably three full years of historical sales data in order to perform historical analysis of a client's sales data in order to calibrate the forecast model. The historical data is used by Analytical Services to formulate their forecasting models.

The historical data load requires the following data from the historical period:

- Merchandise hierarchy
- Location hierarchy
- Calendar
- Items
- Sales/Inventory

All the files except sales/inventory can be one data file for the entire historical period. The sales/inventory data should be provided as a separate file for each week of historical data.

In general, when in a production environment, five data feeds are required on a weekly basis: MK, LH, Calendar, Items, and Sales. Markdowns Taken is a

recommended weekly feed. Demand Parameters, Price Ladders, and Seasonalities are also required, but not on a weekly basis. One-time only data feeds, which are required at the beginning of all projects, include cross products information, merchandise levels, and location levels. These are used to define items (the optimization level - a combination of MH and LH) and the complete merchandise hierarchy and location hierarchy. All other feeds are optional, depending on a client's business requirements.

Each client has a specific set of levels that defines the merchandise hierarchy and the location hierarchy for that client. The levels, called nodes, correspond to how the client organizes its merchandise or its location. the merchandise hierarchy and the location hierarchy are configuration points.

For example, a client's merchandise hierarchy might be organized as:

- Chain
- Division
- Department
- Class
- Style
- Color
- Size

as shown here:

Division	Department	Class	Style	Color	Size
Women's	Sweaters	Wool	Cashmere Cardigan	Black	Small
					Medium
					Large
				White	Small
					Medium
					Large

and the location hierarchy might be organized as:

- Chain
- Region
- Store

as shown here:

Region	Store
Northeast	New York City
	Boston
	Philadelphia
West	San Francisco
	Seattle
	Portland

MDO defines the root node for the hierarchies. Each node in the hierarchy has only *one* parent node. The intersection of the two hierarchies defines the level of optimization, called the item. For example, the item might be a combination of Style and Region. The optimization level is the same across all merchandise. Items that are grouped into a pricing group are optimized together. MDO aggregates sales data and persists data at the item level.

Required Interface Descriptions

The following section provides a high level description of the required MDO data feeds. Technical details of the standard interface specifications can be found in the MDO *Operations Guide*.

Merchandise Hierarchy

The merchandise hierarchy standard interface describes the organization of the client's retail structure. The merchandise hierarchy is used, in combination with the location hierarchy, as a key configuration point that defines the optimization level.

The merchandise hierarchy is also used:

- to define worksheets, such as at the Department level
- to allow business rules to be assigned at higher levels than the item level
- to aggregate metrics in the application UI and in application reports
- to filter data in the Items Worksheet

Location Hierarchy

The location hierarchy standard interface describes the organization of the client's retail locations. The location hierarchy is used, in combination with the merchandise hierarchy, as a key configuration point that defines the optimization level.

The location hierarchy is used:

- to allow business rules to be assigned at higher levels than the item
- to aggregate sales data to the item level (for example, sales are at store level; items are at region level)
- to aggregate metrics in the application UI and in the application reports
- to filter by location hierarchy in the Items Worksheet

Calendar

The calendar standard interface describes a retailer's fiscal calendar.

The calendar data is used:

- to construct the markdown calendar that defines the valid markdown effective dates.
- to determine in what month the markdown effective date falls. The markdown effective month affects metrics displayed in the application UI and in the application reports such as "Markdown Budget."

Items

The items standard interface describes valid combinations of merchandise and location that specify an item. All items in the system are defined at a single level of the

merchandise hierarchy (typically the lowest level) and a single level in the location hierarchy.

Items are used:

- to define the total set of valid items for markdown optimization. (Some items are typically excluded each week, based on their eligibility.)
- to define key fields that affect the determination by the application of the item's seasonality and model start date. (The model start date defines the date when sales are included in the calculation of forecasts and markdowns.)

Sales

The sales standard interface describes weekly sales, inventory, and order data at the lowest level of the merchandise and location hierarchy. If items are defined at higher levels in either hierarchy, then Markdown Optimization aggregates the data in the sales file to the level required by the items.

The sales data is used:

- to define the current selling price for an item in the absence of any promotions.
- to determine, in combination with analytical parameters, the base demand for an item.
- to calculate a number of historical sales and inventory-related metrics.
- to help define the total inventory to clear through markdown optimization (inventory on-hand is always part of the total inventory to clear and units on-order are typically part of the total inventory to clear provided by the model).

Markdowns Taken

The markdowns taken interface describes permanent markdowns, past, present, or future, that have been entered into a retailer's price change execution system.

The markdowns taken data is used:

- to determine the markdowns that have already been executed in the store and the markdowns that are pending execution in the future. The CURRENT_RETAIL field in the Sales data feed does not reflect pending markdowns; however, the application builds the information about pending markdowns into its forecast. Markdown Optimization will not recommend new markdowns prior to any pending markdowns, but will recommend additional markdowns after pending markdowns if additional markdowns are needed and do not violate business rules.
- to determine the validity of future markdowns based on the number of previous markdowns and the date of the most recent markdown.

Demand Parameters

The demand parameters standard interface describes the mapping between the analytical parameter values generated by Analytical Services and a specific merchandise/location/attribute.

The demand parameter data is used:

- to provide a centralized list for the parameters and their values.

  AS_PARAMETER_ID and AS_VERSION_NUMER are used only by Analytical Services; they are not used by the application.

Price Ladders

The price ladders standard interface describes the price ladders displayed in the application UI.

Price ladders define a client-specific set of markdown prices that can be selected in the application. Prices in the price ladder are expressed either as a price point (PP), as a percentage off the original retail price (PO), or as a percentage off the ticket price (PT). Each of these three types of price ladder can be permanent or temporary. Temporary price ladders are denoted by a t- prefix in the UI.

Seasonalities

The seasonalities standard interface describes the seasonality values (effects related to the time of year) provided by Analytical Services that are used by the application to calculate markdowns and forecasts.

Markdown Optimization uses the seasonalities data in a variety of ways, including:

- To support seasonality searches across the merchandise and location hierarchies.
- The following inference rules are involved in seasonalities:
  - IR_SEASONALITIES – provides the seasonality values to the model from start date to out date.
  - IR_SEASONALITY_ATTRIBUTE – defines the attributes value(s) used for seasonality matching.
  - IR_ITEM_IDS – maps item IDs to seasonality IDs

One-Time Interfaces

One-time global settings are defined only once, at the beginning of a client implementation. For more information, see the Standard Load chapter of the MDO *Operations Guide*.

Cross Products Information

Items are globally defined to be at a specific level of the merchandise hierarchy and the location hierarchy through the cross products interface.

The following list provides details to considering regarding the cross products information data.

- The INTERSECT_NAME is the name of the Key, which defines the purpose or feature for the data, and is either OPTIMIZATION, SALES, WORKSHEET, CLUSTER, or DEFAULT LEVEL. Use the value CLUSTER to enable Flexible Clustering. For more information on Flexible Clustering, see the MDO *Configuration Guide*.
- For each Key, identify the defining level of the merchandise hierarchy and location hierarchy.
- The cross products information is generally loaded only once.
- Sales cannot be loaded until optimization level and the sales level are defined. Worksheets must be defined before an optimization run can occur.

Merchandise Hierarchy Levels

The merchandise hierarchy levels interface is used to specify the names of a retailer's merchandise levels and their order.

The following list provides details to consider regarding the mh levels data.

- The Chain level should always be defined as 1.
- The sequence of level numbers must begin with 1 and increase in increments of 1, without any gaps in the sequence.
- The merchandise hierarchy levels information is generally loaded only once.

Location Hierarchy Levels

The location hierarchy levels interface is used to specify the names of a retailer's location levels and their order.

The following list provides details to consider regarding the lh levels data.

- The Chain level should always be defined as 1.
- The sequence of level numbers must begin with 1 and increase in increments of 1, without any gaps in the sequence.
- The location hierarchy levels information is generally loaded only once.

Standard Load

Markdown Optimization provides two scripts that stage, transform, and load data into the target database tables in the application database. For the data transfer guidelines, see "Data Transfer Guidelines" on page 4-18 in this chapter.

Standard Load Scripts

MDO provides scripts to use for staging and loading the data it requires.

The two scripts are located in %INSTALLATION_DIRECTORY%/modules/tools/bin. The first script, **pl_stage_file.sh**, stages the data from the flat files into the ASH staging tables. The second script, **pl_load_data.sh**, loads the staged data into the application database. These two scripts are used if you need to customize the load dependency tree.

Each script contains options that can be customized. You can customize the options in the following ways (which are listed in order of precedence, with the command line having the highest precedence):

- Using the command line options
- Setting the customization values as environment variables in env.sh
- Setting the customization values in the user's environment

If you do not need to customize the load dependency tree, you can use the following two scripts:

- **pl_stage_client.sh** <*full_path_to_product_directory*> *DatasetFilename*
- **pl_load_client.sh** <*full_path_to_product_directory*>

Sample Load Procedures

Here is a description of the load procedures for the *required* data feeds. The descriptions include the source table and the target table. Refer to the MDO *Operations Guide* for information about the optional load procedures.

Note the relationship between LoadMerchandiseHierarchy, LoadMhTbl, and LoadTCLOSE as well as the relationship between LoadLocationHierarchy, LoadLhTbl, and LoadLTCLOSE.

Load Merchandise Hierarchy

Procedure: com.profitlogic.db.birch.LoadMerchandiseHierarchy

Source Tables:

- ASH_MHL_TBL
- ASH_MH_TBL
- ASH_MH_CDA_TBL

Target Tables:

- MERCHANDISE_HIERARCHY_TBL
- MERCH_ATTR_TBL
- PRODUCT_ITEMS_TBL

Description: This procedure loads the entire merchandise hierarchy, with the exception of the node (CHAIN) that is seeded by the application during installation. It updates the merchandise hierarchy based on the most recent information in ASH_MH_TBL and the levels specified in ASH_MHL_TBL. It completely re-loads MERCH_ATTR_TBL with the most recent data from ASH_MH_CDA_TBL. It also updates PRODUCT_ITEMS_TBL according to the most recent merchandise hierarchy data.

Load Merchandise Table

Procedure: com.profitlogic.db.birch.LoadMHTbl

Source Table: MERCHANDISE_HIERARCHY_TBL

Target Table: MERCHANDISE_TBL

Description: This procedure completely re-loads MERCHANDISE_TBL from MERCHANDISE_HIERARCHY_TBL. MERCHANDISE_TBL is a horizontally flattened view of the merchandise hierarchy, used to improve the performance of other load procedures and the application UI.

Load TClose Table

Procedure: com.profitlogic.db.birch.LoadTCLOSE

Source Tables:

- MERCHANDISE_TBL
- CLIENT_HIERARCHY_LEVELS_TBL

Target Table: TCLOSE_TBL

Description: This procedure completely re-loads TCLOSE_TBL from MERCHANDISE_TBL using merchandise hierarchy levels specified in CLIENT_HIERARCHY_LEVELS_TBL. TCLOSE_TBL is a vertically flattened view of the merchandise hierarchy, containing each merchandise node with all its parents. This table is used to improve the performance of other load procedures and the application UI.

Load Location Hierarchy

Procedure: com.profitlogic.db.birch.LoadLocationHierarchy

Source Tables:

- ASH_LHL_TBL
- ASH_LH_TBL
- ASH_LH_CDA_TBL

Target Tables:

- LOCATION_HIERARCHY_TBL
- LOCATION_ATTR_TBL

Description: This procedure loads the entire location hierarchy, with the exception of the node (CHAIN) that is seeded by application during installation. It updates the location hierarchy based on the most recent information in ASH_LH_TBL and the levels specified in ASH_LHL_TBL. It completely re-loads LOCATION_ATTR_TBL with the most recent data from ASH_LH_CDA_TBL.

Load Location Table

Procedure: com.profitlogic.db.birch.LoadLHTbl

Source Table: LOCATION_HIERARCHY_TBL

Target Table: LOCATION_TBL

Description: This procedure completely re-loads LOCATION_TBL from LOCATION_HIERARCHY_TBL. LOCATION_TBL is a horizontally flattened view of the location hierarchy, used to improve the performance of other load procedures and the application UI.

Load LTClose Table

Procedure: com.profitlogic.db.birch.LoadLTCLOSE

Source Tables:

- LOCATION_TBL
- CLIENT_HIERARCHY_LEVELS_TBL

Target Table: LTCLOSE_TBL

Description: This procedure completely re-loads LTCLOSE_TBL from LOCATION_TBL using location hierarchy levels specified in CLIENT_HIERARCHY_LEVELS_TBL. TCLOSE_TBL is a vertically flattened view of the location hierarchy, containing each location node with all its parents. This table is used to improve the performance of other load procedures and the application UI.

Load Calendars

Procedure: com.profitlogic.db.birch.LoadCalendars

Source Table: ASH_CAL_TBL

Target Table: PERIODS_TBL

Description: This procedure updates the PERIODS_TBL, which is seeded by the application during installation. The following columns in PERIODS_TBL are updated:

- FISCAL_YR
- FISCAL_MO
- FISCAL_WK
- FISCAL_QUARTER
- FISCAL_HALF
- CALENDAR_YR
- CALENDAR_MO
- CALENDAR_WK
- CALENDAR_QUARTER
- SEASON (the rows derived from ASH_CAL_TBL)

Load Items

Procedure: com.profitlogic.db.birch.LoadItems

Source Tables:

- ASH_ITEMS_TBL
- ASH_ITEMS_CDA_TBL
- ASH_CP_TBL

Target Tables:

- ITEMS_TBL
- ITEMS_CDA_TBL

Description: This procedure inserts new data and updates existing data in ITEMS_TBL from ASH_ITEMS_TBL, based on the optimization level specified in ASH_CP_TBL (cross products information). It also inserts new data and updates existing data in ITEMS_CDA_TBL from ASH_ITEMS_CDA_TBL.

Load Sales

Procedure: com.profitlogic.db.birch.LoadSales

Source Tables:

- ASH_SALES_TBL
- MERCHANDISE_TBL
- LOCATION_TBL
- ITEMS_TBL
- ASH_CP_TBL
- PERIODS_TBL

Target Table: ACTIVITIES

Description: The sales data is loaded from ASH_SALES_TBL, which is populated at the sales level for merchandise. This procedure loads data that is aggregated to the optimization level. It is stored in the ACTIVITIES table at the optimization level. Any number of weeks of data can be provided in ASH_SALES_TBL. The load procedure processes one week at a time, inserting new data or updating existing data.

Load Markdowns Taken

Procedure: com.profitlogic.db.birch.LoadMarkdownsTaken

Source Tables:

- ASH_MDTAKEN_TBL
- ITEMS_TBL
- MERCHANDISE_HIERARCHY_TBL
- LOCATION_HIERARCHY_TBL
- PERIODS_TBL

Target Tables:

- HIST_MARKDOWNS_TBL

Description: This procedure compares the records from the current week's ASH_MDTAKEN_TBL feed with the records in HIST_MARKDOWNS_TBL and archives all the matches in HIST_MARKDOWNS_ARCH_TBL. It then replaces the records in HIST_MARKDOWNS_TBL with the new records from ASH_MDTAKEN_TBL.

Environment Customization File

Here is an example of the environment customization file (**env.sh**):

```
#This is the environment customization file.
#Please define all customization values here.

#The mail client and address to send all messages to:
#MAIL=mailx
#REPORT_ADDRESS=error_mail@your_domain.com

#Number of parallel processes to run load procedures:
PARALLEL=2

#Directory with data control files:
#CONTROLDIR=/ASHschema/controlfiles

#Directory to store logs:
#LOGDIR=/tmp/load_logs

#Directory to move old logs to.
#If this variable is not set, the logs will be overwritten.
This folder is not required to exist and will be created at the time
#of archiving the logs.
#
#If all old logs should be preserved, it is possible to
#archive the files into a new unique folder, such as:
#LOGDIR_ARCHIVE=
#/tmp/load_logs/archived_logs_'date +%Y%m%d_%H%M_%S'
#
```

```
#If only the archive of the previous run is important, then
#archive the files into the same folder, such as:
#LOGDIR=/tmp/load_logs/archived_logs

#Number of errors to allow during load
ERROR_THRESHOLD=50
```

Using the Scripts

The **pl_stage_client.sh** script calls **pl_stage_file.sh**. The **pl_load_client.sh** script calls **pl_load_data.sh**.

Usage: pl_stage_file.sh *[OPTION]... [FILE]...*
Loads the files into the database.

Options:

Table 4–2 pl_stage_file.sh Options

-a DIR	--logdir_archive=DIR	directory to archive old log files
-c DIR	--controldir=DIR	directory with data control files
-e NUM	--errorthreshold=NUM	number of errors to allow in load (for DB2, it is a warning threshold)
-l DIR	--logdir=DIR	directory to store logs
-r DIR	--configroot=DIR	configuration root directory
-h	--help	displays help and exits

Usage: pl_load_data.sh *[OPTION]... [LOADPROCEDURE]...*
Runs the load procedures in the database.

Options:

Table 4–3 pl_load_data.sh Options

-a DIR	--logdir_archive=DIR	directory to archive old log files
-e NUM	--errorthreshold=NUM	number of errors to allow in load (overwrites the procedure's default limit)
-l DIR	--logdir=DIR	directory to store logs
-r DIR	--configroot=DIR	configuration root directory
-h	--help	displays help and exits

Errors

The Standard Load verifies the records in each staging table. Each record that fails the verification is removed from the staging table and placed in another table so that the load can continue and so that the failed records can be reviewed.

If a load procedure fails and the threshold is exceeded, you will see the message "The specified error threshold has been exceeded for this load procedure." If this occurs, you should correct the existing data problem and re-run the load procedure as well as any child load procedures.

You can configure the threshold values for error handling in the properties file, **dbError.properties**. The values you set in this file override the corresponding application default values. The default value for the threshold of records failed is 100%. The default value for the total record threshold is 0%. Threshold values are expressed as a percentage. Note that the percentage symbol should not be included. Once you have created this file (which should be stored in **com/profitlogic/db/common/resources/dbError.properties** and called as a argument from there), you need to load it into the database schema using the procedure.

Here is a sample **dbError.properties** file:

```
################################################################################################
#This properties file contains all error customizations
#
#Note:all thresholds should be satisfied in order for the load procedure to succeed
#
################################################################################################
#LoadPromotions error customizations
#
#Total error threshold is set to 0% of all records (default is 0%):
LoadPromotions.total.threshold=0
#
#Threshold of records failed with error 1205 should not exceed 100% (default is 100%):
LoadPromotions.1205.threshold=100
#
#Threshold of records failed with error 1207 should not exceed 100% (default is 100%):
LoadPromotions.1207.threshold=100
#########################################################################
```

In the **dbError.properties** file, you can set the total error threshold as well as a separate threshold for specific verifications. When configuring the error threshold for specific verifications, you use the error message number to indicate which verification you are setting the error threshold for. The sum of all the individual thresholds cannot exceed the total threshold.

Loading the dbError.properties File

Once you have created the **dbError.properties** file, you can load it, as follows:

dbpropertiesinstaller.sh <config_root> conf/com/profitlogic/db/common/resources/dbError.properties, where config_root is the root directory of the Markdown Optimization configuration files.

The format for the file <db_connections_properties> is as follows:

For Oracle:

```
db.type=oracle
db.driver=oracle.jdbc.OracleDriver
db.url=jdbc:oracle:thin:@<db_host>:<db_port>:<db_SID>
db.password=<db_password>
```

where

<db_username> is	the username for the database connection
<db_password> is	the password for the database connection
<db_host> is	the host name of the database server
<db_port> is	the port number of the database server

<db_SID> is	the SID or SERVICE_NAME value for the database from the tnsnames.ora file.

Load Order

The load procedures must be loaded in a specified order. Refer to the MDO *Operations Guide* for details.

Standard Load Steps

Each procedure consists of the following sub-procedures:

1. Setup
2. Pre-load Verification. All n processes are run in parallel.
3. Finish Pre-load Verification.
4. Load. All n processes are run in parallel.
5. Post-load Verification. All n processes are run in parallel.
6. Finish Post-load Verification.
7. Tear-down.

Standard Dataset

The Markdown Optimization standard dataset is a set of raw data provided as flat files with the application that

- is shipped with the application and is copied into the installation directory
- requires the front-end configuration grids in the installation directory
- contains five weeks of data
- contains only valid data, so no validation errors should occur during the standard load
- provides data that is sufficient to verify the installation and initial basic configuration of the product
- does not provide a complete set of data to explore the total functionality of the application
- provides data that can be loaded using the standard load procedures, and, once loaded, can be used to perform a model run
- provides data sufficient to permit the launching of the application and invoke the application UI without any additional configuration after the model run is complete
- does not provide any error conditions
- provides one eligibility query for all the merchandise in the dataset.
- includes the expected results of the optimization run
- supports markdowns and forecasts
- uses the default error threshold settings for data validation (procedures = 0 and specific validations = 100). This permits the dataset to fail on any validation error. All invalidated rows are store in appropriate “BAD” staged table.

- requires that an empty schema be created before the data is loaded (part of the standard installation)
- is staged and loaded using pl_stage_client.sh and pl_load_client.sh. See the beginning of this chapter for details about staging and loading data.
- is validated using published load validations
- imports the attributes such as promotions, markdowns, and business rules that are hierarchical in application of the attributes
- contains no sendback data. To test sendbacks, markdowns must be taken, which are then processed during the next weekly load.

Dataset Data

The data in the dataset consists of 3,107 items and 156 pricing groups and 4,682 items as part of pricing groups. Details about the data characteristics are shown in the following table. For information about the standard load procedure and the target table for each interface, see the beginning of this chapter.

Table 4–4 Markdown Optimization Dataset Data

Data Interface Table	Data Characteristics
Merchandise Hierarchy Levels (ASH_MHL_TBL)	Eight levels: Chain Company Division Department Class Style Color Product Key (Chinese value for)
Location Hierarchy Levels (ASH_LHL_TBL)	Five levels: Chain Company Zone Price zone (Chinese value for) Store
Cross Products Information (ASH_CP_TBL)	MH-LH intersections: Optimization Product key (merchandise level - Chinese value)) Store Worksheet Department (merchandise level) Chain (location level) Sales Product key (merchandise level-Chinese value) Store (location level) Cluster CHAIN CHAIN DEFAULTLEVEL CHAIN CHAIN
Merchandise Hierarchy (ASH_MH_TBL)	34,444 merchandise items at the product key level - Chinese value
Location Hierarchy (ASH_LH_TBL)	6 price zones (Chinese value) 845 stores
Items (ASH_ITEMS_TBL)	7,789 items

Table 4–4 (Cont.) Markdown Optimization Dataset Data

Data Interface Table	Data Characteristics
Sales (ASH_SALES_TBL)	5 weeks of data
Markdowns Taken (ASH_MDTAKEN_TBL)	270 external markdowns
Promotions (ASH_PROMOS_TBL)	5,000 promotions (inclusions set as promo_price at the item level)
Distribution Center Inventory (ASH_DCI_TBL)	5 weeks of data
Distribution Center Allocation (ASH_DC_ALLOCATION_TBL)	8,400 allocations
Demand Parameters (ASH_PARAMETER_VALUES_TBL)	160 elasticity values
Price Ladders (ASH_PRICE_LADDERS_TBL)	178 price ladders
Seasonalities (ASH_SEASONALITY_MAPS_TBL) and (ASH_SEASONALITY_VALUES_TBL)	300 seasonality settings
Pricing Groups (No interface or staging table)	200 pricing groups
Business Rule Instances (ASH_BRM_INSTANCE_TBL)	3,023 instances of default rules: 509 OUT_DT 1,000 INVENTORY_TARGET 1,501 PLANNED_START_DT other

Modifying the Dataset

You can modify the dataset to change or add data. Here is a sample procedure for changing the unit cost for an item.

1. Load the staged files into the ASH tables.

   cd *<install_base>/modules/tools/bin*

   bash ./pl_stage_client.sh *full_path_to_product_ directory COEDataset*

2. Identify the target table for the item you want to modify. You can trace back from the P4P_DISPLAY_ITEMS view to the ITEM_DATA table (which supplies the unit cost data) to the ITEMS_TBL table (which populates ITEM_DATA). So, the unit cost data must be changed in ASH_ITEMS_TBL.

3. Determine (from ASH_ITEMS_TBL) the merchandise key and the location key for the ITEMID whose unit cost you are changing.

4. Create a script to apply the change directly to the ASH stage table.

   Here is a sample script called **pl_update_dataset.sql**.

```
UPDATE ash_items_tbl
     SET unit_cost=16.02
WHERE merchandise_key='10000009'
     AND location_key='5';
COMMIT
```

5. Run **pl_update_dataset.sql**

   using sqlPlus on the target schema.

6. Load the data.

7. To verify that the change was made, view the item in P4P_DISPLAY_ITEMS or in ITEM_DATA.

Sample Model Run Results

Here is a sample of expected model run results using the sample dataset provided with the CD image in an "out-of-the-box" configuration:

```
Stoplight Summary
Yellow     3712
Green       1733
```

Message Breakdown

Stoplight	Category	Resource	Instance Count	Message
Yellow	Item Data	engine.status.didyma.noTicketPrice	146	This item has not Ticket Price
Yellow	Item Data	engine.status.didyma.noStartDate	146	This item has no Start Date
Yellow	Item Data	engine.status.agorai.noCandidatePrices	160	No candidate prices available for markdown: filtered Price Ladder is empty
Yellow	Item Data	engine.status.agorai.effectiveDateMustPrecede OutDate	9	OutDate precedes Effective Date
Yellow	Item Data	engine.status.agorai.badlastStoreCount	523	Last week of historical activity does not have a good Store Count
Yellow	Inactivity	engine.status.agorai.noUsefulSales	2135	Data too dirty to determine demand: zero useful sales found
Yellow	Error	engine.status.didyma.itemBuild	74	Item contains other errors
Yellow	Error	engine.status.didyma.collectionBuild	28	Pricing Group contains other errors
Yellow	Error	engine.status.agorai.forecasterMissing	342	Unable to locate an external forecaster or create an internal forecaster for Item 1023
Yellow	Error	engine.status.agorai.collectionItemErrors	149	Pricing Group contains items with errors
Green	Markdown Blocked	engine.status.agorai.noMarkdownPricesAt EffectiveDate	3	No markdown prices available on Effective Date
Green	Not Recommended	engine.status.agorai.sellsOutWithout Changes	1412	Sells to target without changes; Markdown not recommended
Green	Not Recommended	engine.status.agorai.notRecommended	285	Markdown not recommended for this week
Green	Warning	engine.status.agorai.priceAboveFull	33	A relative price is higher than the full price: 1.001

Data Transfer Guidelines

The standard interface document describes the data requirements of the Price application, but does not describe the mechanism and file naming conventions for transferring data to Oracle during Price implementations. The purpose of this document is to provide additional details about the transfer of data files to and from Oracle. Note that this document is not version specific and applies to all implementations.

This section is intended to provide the common conventions expected across all feed and sendback files.

File Format Conventions

All historic and weekly feed files and weekly sendback files are expected to follow certain common conventions.

- All files should be ASCII text files.
- Every line, including the last one in each file, ends with an EOL (End Of Line) sequence. LF (0x0A) is the preferred EOL sequence, CR LF (0x0A 0x0D) is acceptable but discouraged. Every EOL sequence should be the same.
- Files should not contain any blank lines.
- ASCII character codes used in the files should be limited to the printable characters (0x21-0x7E), Space (0x20), HT (0x09), CR (0x0D) and LF (0x0A). Files may be stripped of any other characters. The use of CRs and ITs is allowed but discouraged.
- Files should not contain other control characters or characters with the high bit set (0x80 and above).
- Multi-byte character sets are outside the scope of this version of the document.
- Each line of a file represents one database record.
- Each line contains one or more data fields separated by the delimiter character.
- Delimiters should occur between each data field in the file. They may also occur between the last data field and the EOL.
- The preferred field delimiter is '|' aka "pipe" or "vertical bar", ASCII character code (0x7C). Other delimiters must be negotiated specially if necessary.
- There should be no extra white space around the field delimiters. The only characters which should occur between any pair of delimiters (or a delimiter and the BOL/EOL) is the data for that field.
- The delimiter character must not occur within data fields.
- Every line in a file should contain the same number of fields and delimiters. Empty (null) fields are represented by two adjacent delimiters with no characters between them.
- 8.The use of fixed width fields with or without delimiters must be negotiated.

File Packaging and Encoding Conventions

Oracle supports PGP encryption of these files for clients that desire additional data security. This format is widely supported. Using encryption is strongly recommended if the files are not being transferred over a VPN or SSL connection. Oracle will provide its current public key upon request via email. The client should also provide their PGP

public key to Oracle and sign the data files with their key. This allows Oracle to verify that the files are intact and have not been altered in transit.

Oracle prefers to receive data files compressed to conserve disk space and improve transfer times. The preferred file compression format is GNU Zip (aka gzipped). This format is widely supported. It is recommended that you use version 1.3.5 or better. Note that since compression is already built into the encryption process it would be redundant in that case.

Oracle prefers to receive individual feed files, however if absolutely necessary files can be bundled together using the POSIX standard "tar" archive format. This format is widely supported. If this is done, Oracle prefers that the encryption or compression be applied after the tar bundle is created. Recompressing or encrypting a tar bundle containing previously compressed or encrypted files is strongly discouraged.

File Naming Conventions

All feed or sendback files are expected to follow a standard naming scheme. Anything outside of this scheme must be specially negotiated in advance. A filename consists of a base name and suffixes indicating it's base type, data set membership (usually a date stamp), and any encoding or packaging which has been applied to it. The scheme is meant to avoid ambiguity about the contents of the files and the order in which any transformations were applied.

Standard base names are listed in the appropriate section for each feed or sendback type.

File names should be all lowercase containing only [a-z], [0-9], underscores, and dot-separators between the suffixes.

Standard file type suffixes:

- .dat suffix for data files
- .tch suffix for "touch" files (see the "Other Files" section below)
- .chk suffix for "manifest" files (see the "Other Files" section below)

Date stamp (numeric) or sequence number suffixes:

- Preferred format #1 YYYYWW (fiscal year and fiscal week). This is the expected format for data feed files.
- Preferred format #2 YYYYMMDD (calendar year, month and day). This is the expected format for sendback files.
- The number of options are restricted here to avoid nasty ambiguities. Other formats such as YYYYJJJ (calendar year and Julian day) are subject to negotiation.
- The date stamp MUST be consistent for all the files in a set.

Standard encoding suffixes:

- Files which been encrypted with PGP/GPG should use .pgp or .gpg.
- Files which have been compressed with gzip should use .gz.
- If it is absolutely necessary to bundle multiple files together this may be done using the standard "tar" format. The bundle should be named for the file set it contains (e.g. "weekly") and use the suffix ".tar". With the date stamp this would give a fully decorated name of "weekly.200535.tar". If bundling is done, it is recommended to bundle first and then compress (.tar.gz) or encrypt (.tar.pgp).

Even if the client OS has severe filename restrictions it is recommended that the standard names still be used via the renaming capability in FTP.

The preferred sequence of decorations is as follows:

- unique base name corresponding to contents, e.g. - item.
- file type suffix, eg. - .dat.
- date stamp for the file set, e.g. - .200535.
- compression or encryption indicator, e.g. - .gz.

A fully "decorated" data file name would look like this:

item.dat.200535.pgp -or- item.dat.200535.gz

A fully decorated check file name would look like this:

weekly.chk.200535.pgp -or- weekly.chk.200535.gz

A fully decorated touch file name would look like this:

weekly.tch.200535

Note that the touch file is not encrypted or compressed.

Other Files

These files are given base names based on the file set that they protect. For example "weekly" for weekly data feed files or “markdown” for markdown sendbacks. The base name to be used is called out in the sections below.

Manifest or Check Files. It is often desirable to provide an additional file that acts as a shipping manifest for a given file set. This file contains a set of records describing the files being sent. It consists of the following fields using the same delimited format as the data files. It does not list itself or the touch file to avoid circular reference issues. The file should be compressed or encrypted like the data files it protects. It typically contains the following columns:

- Filename – this is the name of the data file being described. It should be the name of the uncompressed or unencrypted file since that is the form on which the checks can be carried out.
- Record count – this is the most robust piece of information and should still be accurate after all the processing of the received files is completed. It should correspond to the number of lines in the original uncompressed or unencrypted file.
- Byte count – this should be the number of bytes in the original uncompressed, unencrypted ASCII data file, this may not be accurate after a dos2unix conversion (stripping CRs etc.) but should match before that point.
- Checksum – if possible providing strong checksums such as an MD5 or SHA1 "digest" for every file is encouraged. It should be calculated for the original uncompressed or unencrypted ASCII file. The type of checksum used should be negotiated. For the volumes of data under consideration 16 or even 32 bit check sums are effectively useless and will not be considered. Note that PGP/GPG and GZIP already provide a certain level of integrity checking.

Touch Files are empty files created to indicate that the transfer of all other files in the set is complete. This is done to prevent "race condition" bugs caused by the time gap between when a file is actually created and when it's full contents are available. A touch file with data content is an oxymoron because this reintroduces the race

condition that they're supposed to prevent. Touch files should not be compressed or encrypted for this same reason.

File Transfer Conventions

This section reviews recommended conventions.

- The client is expected to initiate all transfers. Pushing feed files to the appropriate Oracle FTP server, and pulling sendback files from it.
- All FTP transfers should be done in binary mode.
- All transfers should be done in passive mode.
- All files should be transferred in some type of compressed format.

Historical Data Feed

The MDO application requires a one-time feed of historical data with the following requirements:

Prior to Oracle providing the client a dedicated FTP site for weekly production, clients are asked to transfer all data files to ftp.oracle.com. Oracle will provide a client unique user name and password for this server. For security reasons, this information is provided to the client over the phone and no record is kept at Oracle of the password issued. In the event of a lost password, the client should contact Oracle to have it reset. The files should be placed in the /inbound/current directory of the FTP server.

There are five types of data required for the historical data load: Merchandise Hierarchy, Location Hierarchy, Calendar, Items, and Sales / Inventory / Orders. Oracle prefers clients to provide a single Merchandise Hierarchy, Location Hierarchy and Items file for the entire set of historical data, rather than one file for each historical week All historical sales records should have corresponding entries in the Hierarchy and Items files.

The standard base names for the files are:

- mh – merchandise hierarchy
- lh – location hierarchy
- cal – fiscal calendar
- item – item data
- sales – weekly sales and inventory data

Oracle prefers to receive the historical sales data broken down by week, producing files with the same names and formats as the weekly sales file (as described below). If necessary, clients can combine multiple historical weeks into monthly or quarterly sales files.

File names should always have a date stamp that corresponds to the year and fiscal week contained in the file. In the event that a sales file contains more than one week of data, the last (most recent) year and week contained in the file should be used as the date stamp.

Weekly File Transfer Process

Here is an example process for an Oracle-specific weekly transfer of data.

Note: Note that your process will depend on your automation infrastructure.

Data Input Files

The following are the requirements for the weekly transmission of data input files to Oracle:

All files should be pushed via FTP from client's source system to the production database server in the Oracle environment.

All files should be sent to the "/inbound/current" subdirectory of the <Client> FTP account.

Standard base names for the weekly feed are:

- budget – markdown budget
- cal – fiscal calendar
- dca – distribution center allocation
- dci – distribution center inventory
- item – item data
- item_cda – item client defined attributes
- lh – location hierarchy
- lh_cda – location hierarchy client defined attributes
- lh_rename – location hierarchy changes
- md_taken – markdowns taken outside the system
- mh – herchandise hierarchy
- mh_cda – merchandise hierarchy client defined attributes
- mh_rename – merchandise hierarchy changes
- promo – planned promotions
- sales – weekly sales

Oracle prefers that these files use the "year and fiscal week" style datestamp. This means that prior to encryption or compression the data file names should follow the pattern:

name.dat.YYYYWW

name corresponds to type of data, e.g., "sales", "mh", "cal".

YYYY is the 4-digit year

WW is the 2-digit fiscal week

For example:

Weekly sales date for fiscal week 5 of year 2002 would be: sales.dat.200205

Markdown budgets for fiscal week 37 of year 2003 would be: budget.dat.200337

The weekly files should include a manifest file as described above, using the base name "weekly".

Name: weekly.chk.YYYYWW

The weekly files should also include a touch file as described above, using the base name "weekly". This is an empty (zero bytes) file whose transfer indicates that all other files (including the record count file) for the current week have been successfully transmitted

Name: weekly.tch.YYYYWW

The following list is an example of a typical set of weekly inbound data files:

File Name	File Contents
cal.dat	Fiscal year calendar
mh.dat	Merchandise hierarchy
lh.dat	Store hierarchy
sales.dat	Sales and inventory by store
items.dat	Item definitions
budget.dat	Markdown budgets
md_taken.dat	Taken markdowns
promo.dat	Current promotional prices
dci.dat	Warehouse inventory
weekly.chk	Summary information (record counts) for verifying the file transfer.
weekly.tch	Touch file that indicates all weekly files have been transferred.

Weekly Schedule: Weekly batch processing can begin only when all data files, the check file, and touch file have been received on the Oracle database server. If the files have not arrived by the designated <Inbound Late Time> an email notification will be sent. If files have not arrived by the designated <Inbound Cutoff Time> the weekly batch process may not be completed on schedule. Support and escalation procedures will be defined in the event of inbound file delays.

Data Output Files (Sendbacks)

Note that this section describes a typical data output sendback procedure. This process requires further definition for integration to the client's price management system.

The MDO application is capable of generating many different forms of output ("sendback") data for transfer to the client's source system. The timing is highly configurable as is the output format and both are client-specific. There may be several different sets; these should be named appropriately.

Each sendback transfer will contain at least one data file. For example, an accepted markdowns file may be named markdowns.dat.YYYYMMDD (where YYYYMMDD is a date stamp corresponding to the day when the file was generated.) All data files in a given set must have the same date stamp

Each sendback transfer will also generate a check file, and a touch file, named after the sendback file set being transferred. These will have identical date stamps to the data files. Example names for the accepted markdowns sendback might be markdowns.chk.YYYYMMDD and markdowns.tch.YYYYMMDD

At the specified time each week, Oracle generates the files and place them in the "/outbound/current" directory of the FTP server. The client should poll for the existence of the touch file and should initiate the transfer once the touch file has been detected.

Email notifications should be configured to confirm completion of the file transfers.

In the case where there are multiple data files to be sent, for example "outdates" and "markdowns", and the sendbacks happen on Tuesdays, Oracle's process is to place the first set of sendback files at a client's specified time in the outbound/current directory. After a predetermined set of time, agreed upon by the client and Oracle, the directory will be cleaned, and the second set of sendback files will be placed.

5

Business Rules

This chapter contains:

The chapter/section on the business rules provides detailed information about questions clients may ask about configuration as well as high level information about how to approach configuring the BRM. See the MDO *Configuration Guide* for technical configuration details, such as using the rule_definition.xml file (the editable file to define business rule values) and brmadmin.sh (used to load the business rules into the application).

Business Rules

The Business Rule Manager (BRM) is a Markdown Optimization utility that is used to view and change business rule settings. Business rules determine which data is used by the application for an optimization. In effect, business rules specify client constraints that are used by the application to determine markdowns and forecasting.

The Markdown Optimization business rules are implemented through the inference rules, using values managed in the BRM. Both the inference rules and the business rules are points of customization for the application.

Business rules restrict markdown recommendations by setting limits on such retail questions as: Is a markdown allowed? What markdown prices are allowed? and What are the inventory goals?

The application provides a file that contains the business rule definitions. The business rule definitions specify the constraints that apply to business rule instances (mappings between location and merchandise hierarchy levels and business rule values). The definitions are configurable; however, most of the business rules have default values that can be used to perform any initial application work.

The following table lists all the business rules available in MDO as well as the default value for each.

Table 5–1 Default Business Rules

Business Rule Name and UI Display Name	Business Rule Description	Default Value
NO_TOUCH_AFTER_LAND No Touch 1st	The minimum number of weeks after the model start date before the item is eligible for a markdown.	7
NO_TOUCH_AFTER_MKDN No Touch Between	The minimum number of weeks between markdowns (after the first one). Note that 0 is not a valid value.	7
MAX_MKDN_NO Max #	The maximum number of markdowns permitted for an item during its entire life cycle.	3
MIN_FIRST_MKDN Min Initial	The minimum amount for the first markdown, which is the lowest percentage drop allowed from the current ticket price at the time of the initial markdown.	0
MIN_OTHER_MKDN Min Other	The minimum amount for any markdown after the first one. The lowest percentage drop allowed from the current ticket price at the time of the markdown.	0
MAX_FIRST_MKDN Max Initial	The maximum amount for the first markdown, which is the highest percentage drop allowed from the current ticket price at the time of the initial markdown.	1
MAX_OTHER_MKDN Max Other	The maximum amount for any markdown after the first one. The highest percentage drop allowed from the current ticket price at the time of the markdown.	1
PLANNED_START_DT Start Date	The date when an item will first be sold.	-
OUT_DT Out Date	The date planned for the end of inventory or by which a specific sell-through target is to be reached.	-
INVENTORY_TARGET Sell Thru %	The planned percentage of sell-through for an item at the outdate. Expressed as a value between 0 and 1.	1

Table 5–1 (Cont.) Default Business Rules

Business Rule Name and UI Display Name	Business Rule Description	Default Value
SALVAGE_WITHIN_TARGET Salv Within	The salvage value of remaining items if the target inventory is met. This is a percentage of the full price. Expressed as a value between 0 and 1.	1
SALVAGE_ABOVE_TARGET Salv Above	The salvage value of the remaining items if the target inventory is above the expected amount. This is a percentage of the full price. Expressed as a value between 0 and 1.	0
NO_TOUCH_BEFORE_OUT (Administrative Business Rule) No Touch EOL	The number of weeks before the outdate when markdowns are no longer permitted.	14
MIN_MKDN_FROM_FULL (Administrative Business Rule) Min % of Full	The minimum markdown, expressed as a percentage of the original full retail price. Used to narrow the list of possible prices for optimization.	0
MAX_MKDN_FROM_FULL (Administrative Business Rule) Max from Full	The maximum markdown, expressed as a percentage of the original full retail price. Used to narrow the list of possible prices for optimization.	1
MKDN_DAY_OF_WEEK Administrative Business Rule) Day of Week	A global setting for the day of the week on which markdowns occur. (Sunday = 1, Monday = 2, Tuesday = 3,....)	2
TEMP_MARKDOWNS_BLOCK	Defines whether TEMP markdowns are counted during the enforcement of the MinMarkdownInterval and MaxNumber-Markdowns (IR_BUSINESS _POLICY). When value is 1, TEMP markdowns count.	1
POS_MARKDOWNS_BLOCK	Defines whether POS markdowns are counted during the enforcement of the MinMarkdownInterval and MaxNumber-Markdowns (IR_BUSINESS _POLICY). When value is 1, POS markdowns count.	1

Table 5–1 (Cont.) Default Business Rules

Business Rule Name and UI Display Name	Business Rule Description	Default Value
MSD_FORCED_START_DT Forced Model Start Date	This is only used if the MODEL_START_OPTION in IR_MODEL_START_OPTION is set to sellThrough. (See the Inference Rule chapter for more information.) The value is an override date that forces the Model Start Date to be the first fiscal day of the week of the Forced Model Start Date.	NONE
MSD_SELLTHROUGH_PCT Model Start Sell Through Pct	This is only used if the MODEL_START_OPTION in IR_MODEL_START_OPTION is set to sellThrough. (See the Inference Rule chapter for more information.) The value is a threshold that triggers the assignment based on a ratio of sold units to total inventory.	2.00%
MSD_MAX_DELAY_WKS Max Model Start Delay	This is only used if the MODEL_START_OPTION in IR_MODEL_START_OPTION is set to sellThrough. (See the Inference Rule chapter for more information.) The value is the maximum number of weeks to wait before automatically assigning the Model Start Date.	2

Notes about Specific Business Rules

This section contains configuration details about specific business rules.

No Touch After Model Start Date

This business rule defines how soon a markdown can be recommended once an item has started selling. An item is eligible for a permanent markdown as early as the first week after the no touch after land period. MDO does not make any recommendations effective before the end of the no touch after land period. Use this business rule to let an item become available in all stores across the merchandise hierarchy level relevant to the item.

No Touch Period Between Markdowns

This business rule defines the number of week that must occur between markdowns for a given item. Once a markdown recommendation for an item is accepted, MDO will not make any further recommendations effective until after the end of the no touch between markdowns period. Since this business rule limits the potential number of recommendations that MDO makes every week, it limits thew number of items that the user needs to evaluate every week.

Maximum Number of Markdowns

This business rule defines the maximum number of times that an item can be taken to markdown by MDO over that item's life cycle. The MDO model constrains its pricing strategy according to this rule. Once an item has actually been taken to markdown the number of times specified by this rule, then the item will no longer be eligible for further markdown recommendations. The use of this rule can reduce potential store work-flow for an item and reduce potential the time a planner takes to review weekly recommendations.

Minimum Amount for the First Markdown

This business rule defines the lowest percentage of the initial ticket price that is permitted as an item's first markdown. MDO will not recommend a first permanent markdown for an item that is less than this percentage value. Users can choose a markdown that does not conform to this business rule.This may force the model to take a deeper markdown later in the item's life cycle. Use this business rule to reduce the number of items the user need to review price changes for and so stores do not have to re-ticket merchandise for small price changes.

Minimum Amount for Subsequent Markdown

This business rule defines the lowest price reduction (expressed as a percentage of the current ticket price) that is permitted for any markdown after the initial markdown for an item. MDO will not recommend a permanent markdown for an item that is less than this percentage value. Users can choose a price that does not conform to this business rule. This may force the model to take more frequent, less profitable markdowns later in the item's life cycle.Use this business rule both to reduce the number of items the user need to review price changes for and so stores do not have to re-ticket merchandise for small price changes.

Maximum Amount for the First Markdown

This business rule defines the highest percentage of the initial ticket price that is permitted as an item's first markdown. MDO will not recommend a first permanent markdown for an item that is greater than this percentage value. Users can choose a markdown above the value for this rule. This may force the model to take a more frequent, less profitable markdowns later in the item's life cycle. Use this business rule to give an item a chance to sell.

Maximum Amount for Subsequent Markdowns

This business rule defines the highest price reduction (expressed as a percentage of the current ticket price) that is permitted for any markdown after the initial markdown for an item. MDO will not recommend a permanent markdown for an item that is greater than this percentage value. Users can choose a markdown above the value for this rule. This may force the model to take a more frequent, less profitable markdowns later in the item's life cycle. Use this business rule both to give an item a chance to sell.

Planned Start Date

This business rule is defined by the client and specifies the date that the MDO uses to specify when an item's life cycle begins and can be used as the model start date if necessary. MDO will not use any data for the period before this date when making markdown recommendations. If you set this date too early then the model may misdiagnose poor sales for an item. If you set this date too late then the model will ignore pertinent sales data. This business rule lets an item be fully set on the floor before MDO analyzes sales and inventory data.

Exit Date

This business rule set the date that the client plans to stop selling an item. This business rule interacts with the inventory target business rule. Setting the exit date close to the current date results in a more aggressive clearance pricing strategy than setting the exit date farther in the future. MDO will not consider any item that does not have an exit date for a markdown recommendation. The exit date can be used to control inventory. Poor-selling items can be priced to meet sell-through targets. Aging styles can be liquidated to make room for new merchandise. The exit date is typically set by model start month or client-provided season code. Historical data can be use to help determine what the exit date should be.

Inventory Target

This business rule defines the amount of inventory that should exist in a store as of the exit date. This business rule interacts with the exit date business rule. If no parameter is set for this rule, then the model assumes that the sell through target at exit date is 100% and executes a price strategy to clear all merchandise by that date. This business rule also interacts with the Salvage Cost business rule and the price elasticity demand parameter. If the value from the salvage cost is greater than the value derived from attempting to meet the sell through target. The model does not attempt to reach the sell through target. Instead, the model tries to achieve a higher margin by salvaging the merchandise. This business rule can be used for merchandise strategy.

Salvage Cost

This business rule defines the known or estimated residual value of merchandise that remains at the end of the life cycle. It is the lowest price point that a retailer selects for selling an item. The model uses an item's residual value to set a floor on possible price recommendations. A higher residual value causes the model to use a less aggressive pricing strategy for clearing inventory by the exit date. A residual value of zero causes the model to consider inventory worthless after the exit date. Perishable or seasonal items have no residual value beyond their exit date, so it is better to sell as much of the merchandise as possible. Other items may have a recovery value that should be taken into consideration when choosing the lowest price point.

Salvage Above Target

This business rule defines the salvage value of the remaining items if the target sell through % is not reached. This value is expressed as a percentage of the full price.

No Touch Before Outdate

This business rule defines the number of weeks relative to the outdate after which markdowns are no longer permitted.

Minimum Markdown as % Full Retail

This business rule defines the minimum markdown permitted, expressed as a percentage of the original retail price. It is used to reduce the number of possible prices considered during the model.

Maximum Markdown as % Full Retail

This business rule defines the maximum markdown permitted, expressed as a percentage of the original retail price. It is used to reduce the number of possible prices considered during the model.

Markdown Day of Week

This business rule defines the client-specified day when markdowns are taken.

Temporary Markdowns

Has a defined end date. MDO assumes that all markdowns are permanent. Temporary markdowns are used because the markdown is expensive. it permits the client to evaluate the financial impact of spreading the cost over the remainder of the item's selling life.

POS Markdowns

Taken at the sales register. Taken on the ticket price and not on the original retail price. Used to manage markdown budgets.

Model Start Date

This section includes Forced Model Start Date, Model Start Date Sell Through Percent, and Model Start Date Delay. The model start date is the date on which an item is considered by the model to be available for sale. Sales on and after this date are considered by the model when calculating markdowns and forecasts. Three business rules are concerned with the model start date. The Forced Model Start Date value forces the model start date to be the first fiscal day of the week. The Model Sell Through Percent value assigns the date based on the ratio of sold units to total inventory. The Model Start Date Delay defines the maximum number of weeks to wait before assigning the model start date. If the sell through option in IR_MODEL_START_OPTION is used, then these three business rules must be configured.

The model start date affects base demand estimation, no touch after land, exit date if expressed as a function of the model start date, and seasonality assignment. If seasonality attribute is a function of start date, then the start date becomes important for demand parameter calculation from historical data and mapping items to these parameters.

Setting the model start date too early could result in a misdiagnosis of poor sales for an item. Setting the start date too late can cause the model to ignore pertinent sales data. The model start date allows an item to be fully set on the floor before the model analyzes sales and inventory data.

Configuring Business Rules

The default settings, are, in general, sufficient for an initial model run. These default values are set at the highest level for everything in the system. The exceptions are Outdates and Planned Start Dates, which are installed without default values assigned. Prior to the model run, you should enter Outdates, using either the BRM or through the application UI (Maintaining Merchandise). If you are going to use Planned Start Date as your Model Start Date, you should enter that value as well, using either the BRM or through the application UI (Maintaining Merchandise). (Note that set-plannedstartdate in p4pgui-config.xml must be set to true in order to configure the Planned Start Date via the application UI. Excluded days for the planned start date can also be configured in p4pgui-config.xml.)

You can perform an initial model run using the default values provided with the application. This will allow you to get the system up and running. It will also provided you with a baseline configuration that you can use when planning your advanced configuration. The advanced configuration is necessary in order to obtain meaningful markdown recommendations and forecasts from Markdown Optimization. configuration. The values you assign to the business rules should reflect your business constraints.

In order to do a model run, you must configure the business rule definitions and load them into Markdown Optimization. The default business rule definitions are

contained in /modules/tools/conf/DefaultRules/rule_definitions.xml. An editable copy of the business rule definitions can be found in config/businessrulemgr/rule_definitions.xml. Once you have edited this file, you can use /modules/tools/bin/brmadmin.sh to load the file into the application.

The business rules are implemented through the inference rules, using the custom values set in the BRM. The IR_BUSINESS_POLICY inference rule provides business constraint values used by the model, which it looks up in the BRM. Other inference rules also look up values from the BRM.

6

Inference Rules

This chapter contains:

Introduction

The chapter on the inference rules provides detailed information about questions clients may ask about configuration as well as high level information about how to approach configuring the IRs. See the MDO *Configuration Guide* for technical details about each inference rule.

Inference Rules

Inference rules are SQL queries into the database. Inference rules correspond to specific business policies and are customization points for MDO. For example, views define relevant dates and data, the values needed for model runs, and which metrics to calculate and populate in the tables visible through the UI.

Inference rules define the interface between the data and the model. They provide data for the model run, are used to calculate Key Performance Indicators (KPIs), and are used to configure the UI. All data that is passed to the model is controlled by inference rules. In addition, much of the data that is passed to the ITEM_DATA table and the UI is also controlled by inference rules. (However, some data is passed to the UI directly through the load statements.) The model is configured primarily through ir.sql and load.statements.sql.

Markdown Optimization is installed with default inference rules, provided in the **ir.sql** file, which is located in **config/db.config**. The ir.sql file is overwritten during each subsequent installation. If you are going to customize **ir.sql**, it is recommended that you create a copy of the changes in **config/db.config**. Keeping a copy of your customization can be helpful in troubleshooting. In addition, this will allow you to apply your changes to any upgrade, which is important, as the default inference rules can change between releases of the application.

Two scripts are available that you can use to apply the **ir.sql** to the database schema:

- **plconfiguredb.sh**, used by the installer
- **configdb.sh**, located in **config/db.config**

For example:

$ **bash configdb.sh** *dbalias username password ir.sql*

You can use **configdb.sh** to apply your **custom_ir.sql** to the database.

Note that certain inference rule values can be managed via the Business Rule Manager (BRM).

Notes about Inference Rules

This section describes the inference rules that are most commonly configured during implementations. Additional inference rules and their descriptions can be found in the MDO *Configuration Guide*. The inference rules discussed in this section are:

- Eligibility Filter (IR_ELIGIBLE)
- Worksheet Definition (IR_WORKSHEET_ID)
- Forecastable Inventory (IR_WAREHOUSE)
- Markdown Calendar (IR_MARKDOWN_CALENDAR)
- Promotions (IR_PLANNED_PROMOS)
- Price Ladders (IR_PRICE_LADDER)
- Effective Date/Exit Date (IR_ITEM_DATES)
- Analytical Settings (IR_ITEM_PARAMETERS)
- Model Start Date (IR_MODEL_START_OPTION)
- Pricing Groups (IR_ITEM_COLLECTION)
- Business Policy (IR_BUSINESS_POLICY)

Eligibility Filter

Eligibility defines the criteria that must be met for an item to be loaded into the front end application. The eligibility filter reduces the number of old, new, or meaningless items that the user may have to examine in order to complete actions. Eligibility may be defined by the time since the exit date or by the threshold # units on hand. Typical filtering includes:

- Item (Style-color/Zone) OH > 0 or ST > 98% or Price already at lowest value in price ladder
- Str OH > 0
- No record on weekly sales (including sales and inventory date) interface file

The MDO application supports the ability to specify an eligibility filter (query) to reduce the number of items that are run through the optimization model each week and passed to UI for display to users. This step populates the item_data table, which is the driving table for the Model Run. Eligibility is configured via /database/db.config/load_Statements.sql and isc_ir_eligibility inference rule. Load_ Statements contains an insert into internal_item_data_Tbl, which then populates item_ data.

Oracle's standard practice is to modify this insert statement to query from isc_ir_ eligibility inference rule (isc - implementation services customization) and then configure isc_ir_eligibility to only return items eligible for the ModelRun.

Example

A standard configuration logic is using the item's sales or inventory in the most recent week, the item's exit date business rule, or any item attribute that is in the weekly data feed. For example, if eligibility is defined as an item having at least one record in the last sales or inventory file and on_hand value > 0, then create the eligibility inference rule as follows:

```
CREATE OR REPLACE VIEW ISC_IR_ELIGIBILITY AS
select i.item_id as item_id
from items_tbl i,


   activities a,


db_last_actual_mv mv
where i.item_id = a.item_id
and a.calendar_dt = mv.db_last_actual_dt
     /* At least one record in sales or inventory file, or store OH > 0 */
and (nvl(a.inventory_units,0) > 0
OR nvl(a.net_sales_units,0) > 0
OR nvl(a.store_count_with_inv,0) > 0);
```

In the example above, items_Tbl contains all available items, activities contains all the sales for each item, and db_last_Actual_mv contains one record that defines the last available sales data in the database. So, only items for last week of sales that fit eligibility criteria are present in this inference rule.

As a next step, the load statements need to be modified such that the internal_item_ data_tbl is populated with a query using the isc_ir_eligibility rule. Modify the code in load_statements.sql as follows (or better, create a custom_load_statements.sql):

```
DELETE load_sql_stmt_tbl
WHERE proc_name = 'FELOAD'
     And stmt_execution_order = 100;


DECLARE
     Myclob clob default


INSERT INTO internal_item_data_tbl
    (item_id, pi_id, merchandise_id, location_id,
     hierarchy1_id, hierarchy1_name,
     hierarchy2_id, hierarchy2_name,
     hierarchy3_id, hierarchy3_name, .......
    )
SELECT
     item_id, pi_id, merchandise_id, location_id,
     hierarchy1_id, hierarchy1_name,
     hierarchy2_id, hierarchy2_name,
     hierarchy3_id, hierarchy3_name……
FROM (
SELECT /*+ index(i) full(m) full(l) */
     i.item_id, h.pi_id, h.merchandise_id, h.location_id,
     h.hierarchy1_id, h.hierarchy1_name,
     h.hierarchy2_id, h.hierarchy2_name,
     h.hierarchy3_id, h.hierarchy3_name,……
       FROM isc_ir_eligibility i
INNER JOIN ir_location_hierarchy h on i.item_id = h.item_id
INNER JOIN
```

```
 (SELECT /*+ parallel(i) parallel(c) */ NVL(parent_collection_id,-1) parent_
collection_id,
       NVL(collection_id,-1) collection_id,
       i.item_id
     FROM ITEMS_TBL i left outer join
       (SELECT c.merchandise_id, c.location_id, c.collection_id, ct.parent_
collection_id
     FROM collection_maps_tbl c
       inner join collections_tbl ct on c.collection_id = ct.collection_id
     WHERE ((select chain_flag from ir_item_collection_option)=''Y'' and
ct.parent_collection_id IS NOT NULL)
      OR (select chain_flag from ir_item_collection_option)=''N'') c
ON  (i.merchandise_id = c.merchandise_id AND i.location_id = c.location_id)
 )  eligible on i.item_id = eligible.item_id
);
BEGIN
INSERT INTO load_sql_stmt_tbl
(STMT_SQL, PROC_NAME, STMT_EXECUTION_ORDER)
END;
VALUES (myclob, 'FELOAD', 100);
```

Worksheet Definition

The MDO application groups items into worksheets for users to review and act on markdown recommendations. Worksheets are defined based on the Merchandise and Location Hierarchy. There is a maximum of four hierarchy levels that can define a worksheet. A worksheet must be defined uniquely, using IR_WORKSHEET_ID.

Example

If the first four levels in the merchandise hierarchy are: Company, Division, Group, and Department, and worksheets are defined at Department/Chain level, then create the following view, where hierarchies 1,2,3 and 4 correspond to the merchandise hierarchy levels.

```
CREATE OR REPLACE VIEW IR_WORKSHEET_IDS
(HIERARCHY1,
HIERARCHY2,
HIERARCHY3,
HIERARCHY4,
DESCRIPTION,
MERCHANDISE_ID,
LOCATION_ID,
COLLECTION_FLAG)  AS
select distinct
     hierarchy1_id as hierarchy1
    ,hierarchy2_id as hierarchy2
    ,hierarchy3_id as hierarchy3
    ,hierarchy4_id as hierarchy4
    ,hierarchy4_name as description
    ,m.merchandise_id
    ,l.location_id
    ,'n'as collection_flag
from item_data_cache i
    ,tclose_tbl m
    ,ltclose_tbl l
    ,(select * from ash_cp_tbl where intersect_name = 'WORKSHEET') a
```

```
where m.item_merchandise_id = i.merchandise_id
    and l.item_location_id = i.location_id
    and a.merchandise_level = m.merchandise_level_desc
    and a.location_level = l.location_level_desc
```

Forecastable Inventory

The MDO model considers the following markdown recommendations and forecasts when forecasting and recommending markdowns:

- Store On Hand
- Store On Order
- DC On Hand
- DC On Order

If all four inventory components are included in the model's forecast, then no configuration is required. To remove one or both components of DC inventory, the IR_Warehouse inference rule needs to be configured. The information is then passed to the model via IR_ACTIVITY_DATA.

Example

Here is an example in which the warehouse on order is excluded from the forecast:

```
CREATE VIEW IR_WAREHOUSE
(
     ITEM_ID,
     CALENDAR_DT,
     WAREHOUSE_UNITS,
     WAREHOUSE_ON_HAND,
     WAREHOUSE_ON_ORDER
) AS
select ITEM_ID,
     CALENDAR_DT,
     WAREHOUSE_ON_HAND  as WAREHOUSE_UNITS,
     WAREHOUSE_ON_HAND,
     WAREHOUSE_ON_ORDER
from ir_warehouse_cache_tbl
```

The standard configuration includes all the buckets; however, if a client does not want to include warehouse units in the forecast, then warehouse_units can be modified to exclude a bucket.

NOTE: This view is also used to populate the front end warehouse columns; therefore, if some portion of the warehouse inventory is not forecastable, only WAREHOUSE_UNITS should be configured.

Markdown Calendar

The IR_MARKDOWN_CALENDAR is used to define specific weeks during which no markdowns can be taken. The blackout period rule is similar to the no touch period rule. The MDO model does not make any markdown recommendations effective during a defined markdown blackout period. If a markdown is warranted during the blackout period, the MDO model will adjust the life cycle pricing strategy to make the recommendation effective either before or after the period, depending on which yields the greatest profit. It prevents store personnel from re-pricing merchandise during busy holiday periods and prevent additional markdowns during the defined period.

IR_ FORCED_ MARKDOWNS defines the timing of recommended markdowns. The model recommends a markdown regardless of the forecasted demand. This results in a mandatory clearance of items.

IR_MARKDOWN_CALENDAR

specifies the valid dates for the optimizer to consider markdowns.

This inference rule has the following columns:

- ItemID – the ID of the item being marked down.
- CalendarDate – the date of the candidate markdown. This date should be between the effective date and the outdate.
- Scenario_ID – 0 for optimization run; all other values identify a specific What If scenario.

Example

```
CREATE VIEW ir_markdown_calendar AS
SELECT /*+  first_rows  */
cii.item_id, c.end_calendar_dt+ibp.markdowndayofweek  AS calendar_dt,
cii.scenario_id
FROM  periods_md_cal_tbl c
     ,ir_item_dates cii
     ,ir_business_policy ibp
     ,wif_scenario_tbl overrides
WHERE
  overrides.scenario_id = cii.scenario_id
  AND ibp.scenario_id = overrides.scenario_id
  AND ibp.item_id = cii.item_id
  AND c.end_calendar_dt+ibp.markdowndayofweek  >= cii.effectiveDate
  AND c.end_calendar_dt+ibp.markdowndayofweek  >= cii.startDate +
ibp.daysAfterLand
  AND (overrides.new_blackout_end IS NULL OR c.end_calendar_
dt+ibp.markdowndayofweek  >= overrides.new_blacko
ut_end)
  AND ((cii.outDate - ibp.daysbeforeoutdate ) >= c.end_calendar_
dt+ibp.markdowndayofweek
      OR
     (cii.outDate >= c.end_calendar_dt+ibp.markdowndayofweek  AND
cii.effectiveDate = c.end_calendar_dt+ibp
.markdowndayofweek ))
  AND NOT EXISTS (SELECT 1 FROM ir_markdown_calendar_ex e
     WHERE e.ITEM_ID = cii.item_id
       AND e.CALENDAR_DT = c.end_calendar_dt+ibp.markdowndayofweek );
```

Any dates that need to be excluded from the markdown calendar should be specified in IR_MARKDOWN_CALENDAR_EX.

Example

In order to exclude dates during a specific type of promotion, create the following inference rule:

```
create or replace view ir_markdown_calendar_ex as
select iid.item_id,
     iid.effectiveDate as calendar_dt,
     'Markdown blocked due to promo type ' || trim(pp.promo_type) as reason_key,
     iid.scenario_id
from ir_item_dates iid, planned_promos pp
```

```
where iid.item_id = pp.item_id
  and trim(pp.promo_type) in ('CIRCULAR')
  and pp.start_dt <= trunc(iid.effectiveDate,'D')+6
  and pp.end_dt >= trunc(iid.effectiveDate,'D')
;
```

Promotions

IR_PLANNED_PROMOS defines price recommendation limitations that are caused by future planned promotions. If a price recommendation cannot be below a future planned promotion or no recommendation can be made with an effective date during the week of a promotion, then the MDO model will constrain its recommendations to meet promotional requirements.

The inference rule defines the characteristics of all future planned temporary markdowns and the associated expected lift for each item. In the forecasted range, this is used to determine the current selling price and to implement floor and ceiling restrictions on markdowns. Promotions with lifts are determined based on a historical analysis of an item's demand.

This inference rule has the following columns:

- Item_ID – the ID of the item affected by the promotion.
- Price – the relative price. Price affects demand according to the price effect function.
- Interpretation – The possible values for interpretation are:
- Promo_Floor (2) – a floor promotion.
- Promo_Ceiling (3) – a ceiling promotion.
- Promo_Unrestricted (9) – a promotion that has no restrictions.
- StartDate – the date on which the promotion will begin.
- EndDate – the date on which the promotion will end.
- Priority – a value used to prioritize all the promotions of a given type in order to eliminate any possible conflicts. The default value is 2.

  The actual precedence rules used to determine the promotion used are:

  – 1 – Floor promos win

  – 2 – Lowest price
- Lift – the effect of an external event, such as advertising, on sales when a promotion is in effect. This is used in forecasting. A multiplier applied to the demand.
- LiftType – used to define the lift. The possible values for lift type are:
  – Base (0) – for base media lifts.
  – Relative (1) – for relative media lifts.
  – POS (2) – for percent-off events that are independent of markdown status.
  – Additional (3) – for percent-off events. Applicable only to items that have had one or more markdowns.
  – No_Markdown (4) – for percent-off events. Applicable only to items that have had no markdowns.

  - First_Markdown (5) – for percent-off events. Applicable only to items that have had one markdown.
  - Multiple_Markdown (6) – for percent off events. Applicable only to items that have had two or more markdowns.
- Scenario_ID – 0 for optimization run; all other values identify a specific What If scenario.

Example

```
CREATE VIEW IR_PLANNED_PROMOS
(ITEM_ID, PRICE, INTERPRETATION, STARTDATE, ENDDATE,
PRIORITY, LIFT, LIFTTYPE, SCENARIO_ID) AS
SELECT i.item_id,
     CASE  WHEN pp.promo_pct_off > 0 THEN
        1 - pp.promo_pct_off
       ELSE
        (SELECT ROUND(pp.promo_price/ip.full_price, 6)
         FROM ir_item_prices ip
         WHERE ip.item_id = i.item_id)
     END AS price,
/* interpretation may be configured to one of the types above based on an
attribute or promo_type */
     9 AS interpretation,
     TO_CHAR(pp.start_dt, 'yyyy-mm-dd') AS startDate,
     TO_CHAR(pp.end_dt + 1, 'yyyy-mm-dd') AS endDate,
     2 AS priority,
     1 AS lift,
/* liftType may be configured to one of the types above based on an attribute or
promo_type */
     case when pp.promo_pct_off > 0 then 2 else 0 end as liftType,
     i.scenario_id as scenario_id
   FROM ir_item_dates i,
     planned_promo_maps_tbl pm,
     planned_promos_tbl pp
   WHERE i.item_id = pm.item_id
     AND i.startdate IS NOT NULL
     AND pm.planned_promo_id = pp.planned_promo_id
     AND NVL(pp.promo_excl_fg, 1) = 1
     AND pp.end_dt >= i.startd
     AND (pp.promo_price IS NOT NULL OR
       pp.promo_pct_off IS NOT NULL)
     AND NOT EXISTS (SELECT 1
           FROM planned_promo_maps_tbl m,
             planned_promos_tbl p
           WHERE m.item_id = i.item_id
             AND p.planned_promo_id = m.planned_promo_id
             AND p.promo_excl_fg = -1
             AND p.end_dt >= i.startdate
             AND (p.client_promo_id = '*' OR
              p.client_promo_id = pp.client_promo_id))
```

Price Ladders

Use IR_PRICE_LADDER and IR_PRICE_LADDER_C to configure price ladders. This rule sets the available prices that the model can use for a markdown. These prices are defined relative to the original full retail price and can be trimmed based on specific business constraints.

The model itself enforces the business rules such as Min/Max % Off First and Subsequent Markdown. The view, on the other hand, enforces the Max % Off From Full.

This inference rule has the following columns:

- Item_ID – identifies the item.
- Price – the price relative to the full price for the item.
- Interpretation – Permanent markdown = 1.
- Scenario_ID – 0 for optimization run; all other values identify a specific What If scenario.

When defining price ladders with the client, be mindful of the number of incremental steps on the ladder. The calendar and price ladder are the main drivers on the optimization search space - and too long a calendar or too many price points can impact the model run time. Additionally, there is little value if a price point ladder is in cent increments over the life cycle of an item (for example, there is little difference in $150 and $150.50).

For a price point ladder, consider 15% increments between values or no more than 50 steps for any given item. It is common to suggest that the ladder be denser (more values) at the lower end and gradually increase the distance between points at the higher end. For example, consider $1 increments below $20, $2 below $50, $5 below $75, and $10 below $100.

A common customization is to apply rounding rules. If a percent off ladder is being used in the model, then all the prices can be rounded to a specific ending (such as x.99) or to a price point on another ladder.

Example:

The SQL below provides an example of a price ladder configuration with a simple rounding rule.

```
CREATE VIEW ir_price_ladder AS
 SELECT
     item_id, price, interpretation, scenario_id
    FROM (
      SELECT item_id,
           NVL(ISC ROUND_PRICE(pl.PRICE_LADDER_TYPE,((1-pl.ladder_pct_
off)*ip.full_price),pl.price_ladder_id)/ip.full_price,ROUND(pl.ladder_price /
ip.full_price,6)) AS price,
     1 AS interpretation,
     cbp.scenario_id
   FROM price_ladders pl,
      ir_item_ids i,
      ir_business_policy cbp,
      ir_item_prices ip
   WHERE pl.price_ladder_id = i.price_ladder_id
     AND i.item_id = ip.item_id
     AND cbp.item_id = i.item_id
     AND NVL(ip.full_price,0) > 0
     AND ip.ticket_price >=
          NVL (pl.ladder_price, (1 - pl.ladder_pct_off) * ip.full_price)
     AND NVL (1 - pl.ladder_pct_off, pl.ladder_price / ip.full_price) >
             (1 - NVL (cbp.maxmarkdownpercentoffullprice, 1) )
ORDER BY NVL (1 - pl.ladder_pct_off, ROUND (pl.ladder_price / ip.full_price, 6)
       ) ASC
) ss
```

The ISC ROUND_PRICE function might convert the price to a value that ends in x.99 so that the recommendation from the model conforms to the client convention - 10.99 instead of 11.02 without the rounding.

Effective Date/Out Date

The IR_ITEM_DATES inference rule, used to configure Effective and Exit Dates, defines a set of intervals, beginning with the startdate and ending with the outdate. StartDate is defined as Sunday by default. This view assumes an updated ITEMS_ BRM_RULES table that contains current outdate values. Note that days of the week must be aligned correctly or errors will result.

A common reason to configure this view is to change the effective date from "next week" to "this week" or to obtain the effective date from the markdown calendar.

This inference rule has the following columns:

- ItemID – identifies the item the dates apply to.
- StartDate – the first date that an item is considered to be available for sale. It is not the date on which the item arrives in the store or the date of the first sale. It can be calculated based on sales or it can be supplied directly from the client through a data feed.
- StartSimulationDate – the date on which the simulation starts, which is defined by default by adding one day to the last day of historical activity. The last day of history is always a Saturday, which is the last day that the application has the sales data from the client.
- EffectiveDate – the next valid date for taking markdowns.
- OutDate – the date on which all items are sold or the target inventory value is met.
- DB_Last_Actual_Date – the last day of historical activity.
- Scenario_ID – 0 for optimization run.

Example

The ir_item_dates rule below sets the effective date to nine days beyond the last received sales record.

```
Create or replace view ir_item_dates as
select /*+  first_rows  */
IMR.item_id,
IMR.model_start_dt as startDate,
/* Consider a custom business rule for this week (0) or next week (1) as effective
week */
/* The kit hard-codes a value.  Best Practice is to use the rule from BRM as
described below */
IMR.db_last_actual_dt + (7 * nvl(to_number(brv.isc_mkdn_week),1)) + brv.MKDN_DAY_
OF_WEEK  as effectiveDate,
NVL(ovr.new_out_dt,
(SELECT TO_DATE(TO_SINGLE_BYTE(BRV.out_dt), 'YYYY-MM-DD')
FROM ITEM_BRM_RULES brv
WHERE brv.item_id = imr.item_id)
) AS outdate,
   IMR.db_last_actual_dt + 1  as startSimulationDate,
IMR.db_last_actual_dt,
IMR.item_last_actual_dt,
OVR.scenario_id
 from items_modelrun_tbl IMR,
wif_scenario_tbl OVR
```

Analytical Settings

The IR_ITEM_PARAMETERS inference rule defines the analytical parameters used in a forecast and are provided by Analytical Services.

This view has the following columns:

- ItemId
- GAMMA – price elasticity if PRICEEFFECT is 'power', ignored otherwise
- CRITICALINVENTORY – logic provided by Analytics
- ZEROINVENTORYEFFECT – logic provided by Analytics
- DEMAND_UNCERTAINTY – used only if Model = Stochastic
- MODEL – deterministic or stochastic
- DEMAND_STRATEGY – not case sensitive, must be one of 'Maximum', 'TrimmedMean', 'Trimmedminmean', 'Average', 'Bayesian', 'FullSeason', or 'GenuineBayesian'
- DEMAND_INTERVALS – logic provided by Analytics
- MAXNEWMARKDOWNS – logic provided by Analytics
- ALPHA – Exponential decay parameter, required if DEMAND_STRATEGY is 'Average', 'Bayesian', or 'GenuineBayesian'. Ignored otherwise
- BETA – price elasticity if PRICEEFFECT is 'exponential', ignored otherwise
- PRICEEFFECT – not case sensitive, must be 'power' or 'exponential', other values are considered as 'power'
- INSEASONDISTRIBUTION – Required if DEMAND_STRATEGY is 'GenuineBayesian', ignored otherwise. Not case sensitive, must be one of 'normal', 'poisson', or 'negativeBinomial' if PRIOR_TYPE is 'Normal'; must be 'poisson' if PRIOR_TYPE is 'Gamma'
- INSEASONPARAMETER – Required to specify the width of the distribution if INSEASONDISTRIBUTION is 'negativeBinomial', ignored otherwise.
- USEINTERNALPRIOR – Must be either 0 or 1, and other values are considered as 1. If set to 1, apply the internal prior in addition to the prior distributions available from ir_prior_distribution. If set to 0, only apply the prior distributions if available, otherwise assume prior sigma is infinite and prior mean / prior sigma = 0 (still apply the PRIOR_TYPE).
- INTERNALPRIORBIAS – Required if DEMAND_STRATEGY is 'GenuineBayesian' and the prior distribution is not available from ir_prior_distribution and INTERNAL_PRIOR_MEAN is not available or <= 0, ignored otherwise
- PRIOR_TYPE – required if DEMAND_STRATEGY is 'GenuineBayesian', ignored otherwise. Not case sensitive, must be set to either 'Normal' or 'Gamma' even if the USEINTERNALPRIOR flag is set to 0.
- RELATIVE_INTERNAL_PRIOR_STD – If positive and DEMAND_STRATEGY in ir_item_parameters is 'GenuineBayesian' and the prior distribution is not available from ir_prior_distribution, considered as the override to the relative internal prior standard error that is used to compute the internal prior sigma.Otherwise, ignored and if needed the default value 0.3 will be applied as relative internal prior standard error.

The IR_SEASONALITY_ATTRIBUTE inference rule defines the item attribute value that is used to look up seasonalities. This inference rule has the following columns:

- Item_ID – identifies the item.
- Item_Attribute - the Analytical Services value assigned to the item.tem_ID - identifies the item.

The item_attribute field must match the format of the "attribute_mask" in the seasonality map output from APC MDO.

The PRERUN process will use the view to look up the seasonality for each item and cache it in the items_modelrun_tbl for use during the model run.

```
create or replace view ir_seasonality_attribute
(
     item_id,
     item_attribute
)
as
select i.item_id,
     (select trim(to_char(p.FISCAL_MO,'09')) from periods_tbl p where p.period_
type='FD'
     and p.begin_calendar_dt = i.model_start_dt)
as item_attribute
from  items_tbl i
```

Model Start Date Configuration

The MDO model bases its forecast and markdown recommendations for an item on sales data observed since the item's model start date (items_tbl.model_start_dt). The model start date is typically configured as the date when the sales data for an item should be considered meaningful from a forecasting perspective.

However, the model start date can also be calculated based on sellThru percentage. The standard logic is:

Override model start date using a business rule,

Else calculate model start date based on "2%" SellThrough on "Inventory" (configurable),

Else "3" weeks after 1st sale date (date of first sale after greater of First Receipt date (Items_tbl) and Planned Start Date (business rule).

In order to configure the model start date, the inference rule ir_model_start_option needs to be modified by specifying "sellThrough" as model_start_option. When this option is selected, a value of 'Y' or 'N" must be specified for recalc, use_storeoh_inv, use_store_oo_inv, use_dcoh_inv, and use_dcoo_inv.

```
CREATE OR REPLACE VIEW IR_MODEL_START_OPTION (MODEL_START_OPTION, THRESHOLD,
RECALC, USE_STOREOH_INV, USE_STOREOO_INV, USE_DCOH_INV, USE_DCOO_INV) AS
select 'sellThrough' as model_start_option,
NULL as threshold, --not used for 'sellThrough' option
'N' recalc,  --used to indicate that a recalculation will be used
'Y' use_storeoh_inv, --Store on Hand Inventory
'Y' use_storeoo_inv, -- Store on Order Inventory
'Y' use_dcoh_inv, --DC onHand  Inventory
'Y' use_dcoo_inv -DC onOrder Inventory
from DUAL;
```

This also provides flexibility in choosing which inventory components will be used for calculating the sell-through percentage.

Three business rules need to be configured if "sellThrough" option is specified in ir_model_Start_option: MSD_FORCED_START_DT, MSD_SELLTHROUGH_PCT, MSD_

MAX_DELAY_WK. The model start date is stored in items_tbl.model_start_dt column. The load procedure to calculate and store the value is called from load_statements.

Pricing Groups

Pricing Groups, also known as Collections, are a group of items that are optimized together. Pricing groups in MDO are traditionally managed at the optimization level. MDO also permits pricing groups to be managed at the chain level but optimized at a lower level. Chain level pricing groups can be useful, for example, when adding merchandise to a pricing group in all locations. Instead of adding the merchandise to each location separately, a user can add the merchandise only once, at the chain level, and the merchandise will be added by the application to each location. Although the pricing group is managed at the chain level, the worksheet displays the pricing group at the optimization level to facilitate taking markdowns.

You generally need to change the IR_Item_Collection_Option Chain_Flag from 'N' to 'Y' if the client is editing pricing groups in the MDO front-end and wants to edit pricing groups at the chain level. This flag will trigger the load to create zone level pricing groups from the chain level group. By default, this flag is set to 'N', which indicates that the pricing groups are managed at the level of optimization.

Two inference rules provide configuration points for pricing groups. The IR_ITEM_COLLECTION inference rule is used to provide custom configuration for defining what is included or excluded from the collections load. The IR_COLLECTION_OPTION inference rule contains a flag that is used to indicate whether pricing groups are managed at the chain level or at the optimization level. (Note that IR_COLLECTION_INFO continues to be used for optimization without regard to pricing group management.

Example

```
CREATE OR REPLACE VIEW IR_ITEM_COLLECTION_OPTION (CHAIN_FLAG) AS
SELECT 'Y' AS CHAIN_FLAG FROM DUAL
CREATE OR REPLACE VIEW ir_item_collection
(ITEM_ID, MERCHANDISE_ID, LOCATION_ID
COLLECTION_CLIENT_ID, COLLECTION_DESC)
AS
SELECT item_id,
i.merchandise_id,
i.location_id,
case when iro.chain_flag='Y'then mh1.client_load_id||'/'||i.season_code
else mh1.client_load_id||'/'||i.season_code||'/'||i.location_id end as collection_
client_id,
mh1.merchandise_desc || '/' ||i.season_code as collection_desc
FROM items_tbl i, merchandise_hierarchy_tbl mh, merchandise_hierarchy_tbl mh1, ir_
item_collection_option iro
WHERE mh.merchandise_id = i.merchandise_id and mh1.merchandise_id =
mh.parent_merchandise_id
```

In this example, when the flag is set to 'Y' in ir_item_collection_option, the collection_client_id is set to the parent client load id concatenated with the item's season code. Similarly, the collection_desc column is set to the parent merchandise description concatenated with the item's season code.

Pricing groups can also be created automatically. LoadCollectionsAuto, which is part of weekly load procedures, creates new collections and new collection maps based on the ir_item_collection view. All items with the same collection_client_id are assigned to the same collection. If any item has already been assigned to a collection, or has

already been auto-collected, the load procedure excludes it (via the item_Assigned_Colls_tbl table).

Business Policy

The IR_BUSINESS_POLICY inference rule contains all the business constraints information, such as markdown depth and salvage details, that is used by the model run. This inference rule should produce one row per item to be forecast/ optimized in a model run.

This inference rule has the following columns:

- Item_ID – the ID of the specified item.
- MinMarkdownInterval – the number of days required between markdowns. This is managed by the NO_TOUCH_AFTER_MKDN business rule.
- MinMarkdownPercentOfFullPrice – the minimum markdown, expressed as a percentage of the original full retail price.
- MaxFirstMarkdownPercentage – the maximum amount for the first markdown, expressed as a percentage of the current permanent price (ticket price). This is managed by the MAX_FIRST_MKDN business rule.
- MaxNumberMarkdowns– The total number of markdowns an item can receive during its life cycle. This is managed by the MAX_MKDN_NO business rule.
- NoMarkdownInPromo – a value, not used by default, that can be used by IR_MARKDOWN_CALENDAR or IR_MARKDOWN_CALENDAR_EX to trigger the elimination of markdown dates that are scheduled during a promotion.
- PromoCeiling – Not used by default. The value can be used by IR_PLANNED_PROMOS to affect Promo Type (interpretation).
- InventoryTarget – the number of items expected to remain unsold by the out-of-stock date (also called outdate or exit date). This is managed by the INVENTORY_TARGET business rule as a sell-through percent. The sell-through percent is used to calculate the value for the number of items.
- TargetSellThru – the fraction of inventory that the application should try to sell.
- SalvageValueAboveTarget –- the value of an item when the inventory target is not met, expressed as a dollar amount. This is managed by the SALVAGE_ABOVE_TARGET business rule. The dollar amount is used to calculate the salvage value as a percentage of the full retail price.
- SalvageAboveTargetPercent – the salvage value for unsold items above the sell-through target.
- SalvageValueWithTarget – the value of an item when the inventory target is met, expressed as a dollar amount. This is managed by the SALVAGE_WITHIN_TARGET business rule. The dollar amount is used to calculate the salvage value as a percentage of the full retail price. The value is used by IR_PRICE_LADDER.
- DaysAfterLand – the minimum number of days after the first optimization date before the item is eligible for a markdown. This is managed by the NO_TOUCH_AFTER_LAND business rule. It is used by IR_MARKDOWN_CALENDAR to eliminate some potential markdown dates for optimizations.
- NoMarkdownOnEffective – used by IR_MARKDOWN_CALENDAR or IR_MARKDOWN_CALENDAR_EX to eliminate a specific recommended date as an effective markdown date. This value is not a default value.

- MaxMarkdownPercentOfFullPrice – the maximum markdown, expressed as a percentage of the original full retail price. This is used by IR_PRICE_LADDER to trim the list of candidate prices available to the optimization.
- StockoutLevel – used to determine whether or not the inventory target has been met, for purposes of applying salvage targets. The value is expressed in units and is typically set to 0.
- MaxAbsolutePrice – not implemented. Set to 1.
- MarkdownDayOfWeek– can be used by IR_ITEMS_DATES and IR_MARKDOWN_CALENDAR to indicate the day of the week that is the markdown day.
- DaysBeforeOutdate – used by IR_MARKDOWN_CALENDAR or IR_MARKDOWN_CALENDAR_EX to eliminate a specific recommended markdown date that is close to the outdate. This value is not a default value.
- MinFirstMarkdownPercentage – the minimum amount for the first markdown, expressed as a percentage of the current permanent price (ticket price). This is managed by the MIN_FIRST_MKDN business rule.
- MinSubseqMarkdownPercentage – the minimum amount for every markdown after the first one, expressed as a percentage of the current permanent price (ticket price). This is managed by the MIN_OTHER_MKDN business rule.
- MaxSubseqMarkdownPercentage – the maximum amount for every markdown after the first one, expressed as a percentage of the current permanent price (ticket price). This is managed by the MAX_OTHER_MKDN business rule.
- TempMarkdownsBlock – Used to indicate whether to consider temporary markdowns when calculating MaxNewMarkdowns and when making decisions based on MinMarkdownInterval.
- PosMarkdownsBlock – Used to indicate whether to consider POS markdowns when calculating MaxNewMarkdowns and when making decisions based on MinMarkdownInterval.
- Scenario_ID – 0 for optimization run; all other values identify a specific What If scenario.

Note that this view should not need to be configured, but there are occasional customizations to the "PercentOff" rules or to Salvage rules (for example, calculating Salvage rule as a percentage of unit cost instead of Full Price).

7

Front End Configuration

This chapter contains:

Introduction

There are two main components of MDO configuration: Metrics and Grids. Metrics are the individual data cells within the application that display client data, optimizer results, and dynamic markdown calculations. Metrics are organized through Grids. Grids are configured according to a retailer's requirements to control what metrics are visible within MDO and desired layout.

The chapter on the front end configuration provides detailed information about configuring the MDO UI. See the MDO *Configuration Guide* for additional technical configuration details.

Basics

There are two different file types that comprise the MDO front-end. These include files ending in .xml and .properties. The .xml file controls and manipulates the metrics and data shown in the user interface (UI) while .properties controls labels and a limited set of system functionality configuration. The following sections describe the basic touch points and files that are associated with the front-end configuration.

Configuration Components

The following table identifies the major files used in the UI configuration.

Table 7–1 Configuration Components

Configuration	File Used for Configuration
Standard metrics – all standard metrics included in the application plus up to 24 custom "pass-thru" metrics. Pass-thru metrics are data that is sent to Oracle via the standard data interfaces and is displayed in the applications front-end. No transformations, calculations, or derivations are required.	p4p-column-list.xml
Custom metrics – defines custom metrics available in any grid (or report). Any metric that is not available in the Front-End Metrics document and requires transformation, calculation or derivation is considered custom. If there is any change required to any of the standard metrics, a metric definition needs to go into p4p-custom-columns.xml. Any metric definitions that exist in the custom columns file will override definitions in the p4p-column-list.xml	p4p-custom-columns.xml
Labels and help text – contains predefined placeholders for labels and help text for all columns in all grids (or reports).* The labels and descriptions are generic. The client provides the descriptions that are entered and saved into the file. The application server must be restarted in order to display the updated information.	gridResources.properties
Metrics in the grid – defines the standard and custom metrics available in the grid.	<grid>.xml
Row groupings – defines the row groupings for the grid (how grid data is organized in the grid.	
Price ladder level – define the row grouping level (only one allowed) for price ladder drop-down lists for items worksheet grids.	
Item details – defines the layout of the item details pop-up.	item-details-layout.xml
Views – for items worksheet and merchandise maintenance, defines the available views (grids) for that screen.	p4pgui-config.xml
Reports – MDO reports are defined similarly to grids, but they have some special properties and behaviors.	<report>.xml

Front End Metrics

Refer to the Front-End Configurator for a list of all available front-end metrics. Also, up to 24 custom "pass-thru" metrics are allowed. Pass-thru metrics are data that is sent to Oracle via the standard data interfaces and are displayed in the application's front-end. No transformations, calculations, or derivations are required. Any metric that does not exist in the Front-End Metrics document is considered a custom metric.

In order to create a custom metric, metric name, properties and derivation need to be added to the p4p-custom-columns.xml file. The table below shows different attributes and values for metric definition

Table 7–2 Front End Metrics

Attribute	Possible Values	Description
Label	Defined in gridresource.properties file configuration	This is a reference to a text value that is used as the metric's heading.
Description	Defined in gridresource.properties file configuration	This is a reference to text value that display when mousing over a metric heading.
DB-table-name	P4P_DISPLAY_ITEMS	This is the table or view for MDO to retrieve the metric from.
DB-column-name	Dependent on the columns contained in P4P_DISPLAY_ITEMS	This is the column from the table or view that accesses as its source for data. Is used as an alias when the metric is not available as SQL calculation (derivation) is used in XML.

Table 7–2 (Cont.) Front End Metrics

Attribute	Possible Values	Description
Derivation	Valid PL SQL statement.	Used when calculating a value that depends on multiple columns and logic from P4P_DISPLAY_ITEMS. The derivation attribute is not included when a column exists in P4P_DISPLAY_ITEMS.
Type	Integer, double, number, currency, percent, date, string	Data type of the metric stored in the database. This affects filtering or aggregation that uses this metric.
Display-type	Date, edit, float, integer, currency, percent, static-text, time, button, combo box, drop down, check box, hyperlink, blank	Determines the formatting and display of the metric in the UI.
Read-only-type	Date, edit, float, integer, currency, percent, static-text, time, button, combo box, drop down, check box, hyperlink, blank	Determines the formatting and display in the UI for the metric when screen is in read-only mode.
Filterable	True or False	Determines whether or not the metric is available for filtering in the grids in MDO.
Sortable	True or False	Determines whether or not the metric is available for sorting in the grids in MDO.
Orderable	True or False	Determines whether or not the metric can be placed in a different display order within the grids in MDO.
Hideable	True or False	Determines whether a metric can be removed from the grids in MDO.
Visibility	Visible, not-visible, never-visible	The initial display of the metric in the grids in MDO.
Composable	True or False	Determines if the metric can be used as part of a user-defined metric that is created in MDO.
Filtertype	Text, drop down, date	Defines the data type that is used for filtering on a metric in the grids.
Operatortype	Numeric, list, equals	Defines the operator set that is used for filtering on a metric in a grid.
filter-enum-sql	Any valid SQL statement.	Used in conjunction with the "List" operatortype to define a drop down of preset values to filter on.
GroupID	GROUP_HEADER	Always "GROUP_HEADER"
Function key	P4P_SUM, P4P_MAX, P4P_MIN, P4P_SAME_OR_NULL, P4P_AVG (with or without an argument)	This value represents the method of aggregation used on the grid. An argument is used in P4P_AVG for weighted averages.

Example

When the XML below is added to the p4p-custom-columns.xml, the metric key Cust_PctAllocd can be added to the grid files for display in the MDO UI. Cust_PctAllocd provides the ratio of current units on hand and committed inventory.

```
<column-def>
<key>Cust_PctAllocd</key>
<column-def-properties display-type="integer"
label="p4pgui.Cust_PctAllocd.column.label" type="percent" db-column-
name="XML_Cust_PctAllocd"
description="p4pgui.Cust_PctAllocd.column.description" db-table-
```

```
name="P4P_DISPLAY_ITEMS" derivation="(CASE WHEN
COMMITTED_INV_UNITS > 0 THEN
CURRENT_UNITS_ON_HAND/COMMITTED_INV_UNITS ELSE 0 END)"
sortable="true" orderable="true" hideable="true" expandable="false"
filtertype="text"
operatortype="numeric" groupId="GROUP_HEADER" >
<function key="P4P_AVG">
      <args>INT_INVENTORY</args>
            </function>
</column-def-properties>
</column-def>
```

Note that in order for the metric to display the correct label and description, the following entries needs to be added to the gridResources.properties file.

```
p4pgui.Cust_PctAllocd.column.label= Percent Allocated
p4pgui.Cust_PctAllocd.column.description= Ratio of units on hand and committed
inventory
```

Worksheets and Views

A worksheet is a collection of items. Clients use worksheets for various purposes, such as to view item details, change markdown prices, and accept or decline recommendations. Each worksheet represents a specific department or some other level in the Merchandise Hierarchy. Worksheets are defined at a specific level in the merchandise and location hierarchy (but only up to four levels). The worksheet summary script that comes out of the kit and is used to configure the items worksheet screen is P4p-wksht-summary-grid.xml.

The items worksheet screen contains several standard views that come out of the kit, but the three most commonly used ones are:

- p4p-items-grid-flat.xml – Flat Items view
- p4p-price-groups-grid.xml – Pricing Group (Collections) View
- p4p-loose-items-grid.xml – Aggregation based on Hierarchy View

Additional standard views that come out of the kit are:

- p4p-edit-items-wksht-grid.xml – Edit Worksheet view, identical to an Item Worksheet view, used to configure Add/Remove Items grid
- p4p-edit-group-grid.xml – Edit worksheet view for price groups

Two Maintaining Merchandise Views also come out of the kit:

- p4p-maint-grid.xml – Items view, identical to an Item Worksheet view
- p4p-maint-grid-groups.xml – Pricing Group view, out of Kit (aggregates by pricing group ID)

Maintaining Merchandise views contain additional metrics.

Worksheet summary metrics require more effort and configuration than item worksheet only metrics. Worksheet summary metrics are built on top of item worksheet metrics, which require extra work to aggregate them up to a worksheet level.

In addition to using the p4p-custom-column.xml file to configure client specific metric requirements on the worksheets, the p4pgui-config.xml is used to configure worksheet summary metrics. Three areas are configured that depend on the type of worksheet summary metric.

Example

Used when a metric is defined that will be displayed in the worksheet summary grid:

```
<metric-items name="wkshtRecMDDollars" column-ref="INT_REC_MD_COST"
aggregationType="SUM"/>
```

The name and column-ref attributes refer to two nodes configured in the p4p-custom-column.xml list. This is where they are linked together to aggregate the specific metric (aggregationType) to the worksheet level.

Example

Used for budget metrics that are provided using the standard interface.

These two examples refer to the current fiscal month and next fiscal month budget values. The name attribute refers to a node configured in the p4p-custom-column.xml list while derivation refers to the column in the database where the budget value is found.

```
<metric-m1budget name="m1_budget" derivation="budget"/>
<metric-m2budget name="m2_budget" derivation="budget"/>
```

These two fixed metrics represent formulas that are hard-coded to use defined metric-items with specific name attributes.

```
<metric-fixed name="variance_from_taken_dollars"/>
<metric-fixed name="m1_variance_from_total_cost"/>
```

Example

Used for the footer metrics found on the worksheets.

```
<items-metric id="modified_by" display-name="p4pgui.metrics.modifiedBy.label"
row="2" column="5"/>
<items-metric id="last_modified" display-name="p4pgui.metrics.lastMod.label"
type="date" row="3" column="5"/>
```

The id attribute refers to a defined metric or items with specific name attribute. Also note the attributes that control the column and row that these metrics appear in the footer.

Reports

Markdown Optimization provides standard reports, which users generate and view from the application UI. Standard reports and custom reports can be configured using XML. The generated reports are presented to the user as Excel spreadsheets. Basic formatting of the spreadsheets is defined in the XML configuration file. Report XML files are identified in the config.properties files and are located in the /config/grids directory in the application configuration directory structure. Each report XML file configures a single standard report.

The config.properties file is used to identify the standard reports to be configured using XML. The reportKeys property is a comma-separated list of the properties that are used to specify the name of the XML file for the report.

Example

Here is an example of a sample setup for the config.properties file.

```
# Reports
```

```
reportKeys=sample-md-analysis-report-1,sample-price-changereport-1
sample-plugin-report=sample-plugin-report.xml
sample-md-analysis-report-1=sample-md-analysis-report-1.xml
sample-price-change-report-1=sample-price-change-report-1.xml
```

Example

Here is an example of the structure of an XML report file.

```
report
    worksheet-filter
    page-setup
    column-group-spec
        column
        key
        parent-key
        column-properties
        custom-property
    row-group
        key
        rowgroup-properties
        column-group-override
            column
            key
            parent-key
            column-properties
            custom-property
row-group
```

Standard Reports

Three simple report that use existing metrics defined as part of the front-end configuration are provided with MDO. Each report consist of fewer than 10,000 rows per report.

- Sample-md-analysis-report-1.xml - Provides a report displaying pricing and cost data to aid in markdown decisions
- Sample-plugin-report.xml - Placeholder report to help develop a custom report
- Sample-price-change-report-1.xml - Flat list report outlining pricing data

8

User Management

This chapter contains:

Introduction

User Management is a utility that lets you create, modify, and remove user accounts from a central location. The User Management utility is installed automatically when you install the application.

Each user who accesses the application must have a user account. Each user account is assigned one or more roles that determine the types of functions the user can perform with the application.

Single sign-on is supported so that users can access the entire suite of products, if they are available, without additional authentication.

This chapter provides details about user roles and password policies.

For more information, see the *MDO Configuration Guide.*

About User Roles and User Actions

Roles are defined by a specific set of user actions. The actions that define each role serve to delimit the activities a user can perform. All actions are self-contained. For example, Write does not imply Read. So a role must include all the actions that are necessary for complete functionality. If a role is assigned at a specific level in the hierarchy and that hierarchy level is removed, then the role is removed.

Markdown Optimization comes with a default set of roles, loaded into ROLE_ACTION_TBL. Default action are assigned to the roles. These cannot be deleted. For more information on Business Rule Manager roles and actions and Seasonality Manager roles and actions, see those respective chapters.

The following table lists the MDO roles and the default actions assigned to those roles.

Table 8–1 MDO Roles and Default Actions

Role	Default Action
PRICE_APPROVER	PRICE_APPROVE

Table 8–1 (Cont.) MDO Roles and Default Actions

Role	Default Action
PRICE_SUBMITTER	PRICE_SUBMIT PRICE_COMMENTS_EDIT
PRICE_USER	PRICE_MARKDOWNS_VIEW PRICE_MAINTAINING_MERCHANDISE_VIEW PRICE_BRM_VIEW PRICE_USER_PROFILE_VIEW PRICE_REPORTS_VIEW PRICE_GUARD PRICE_ITEM_INFO_VIEW
PRICE_VIEWER	PRICE_VIEW
BRM_PRICE_EDIT	BRM_PRICE_EDIT PRICE_SEASONALITY_EDIT
BRM_PRICE_VIEW	BRM_PRICE_VIEW PRICE_SEASONALITY_VIEW
BRM_PROFITLOGIC_EDIT	BRM_PROFITLOGIC_EDIT
BRM_PROFITLOGIC_VIEW	BRM_PROFITLOGIC_VIEW
WHAT_IF_SERVICE_USER	MDO_WHAT_IF_SERVICE_EXEC

The MDO roles are defined as follows.

- PRICE_APPROVER – has read-only access to worksheets and can approve submitted worksheets at the specified level in the hierarchy, but cannot submit worksheets.
- PRICE_SUBMITTER – can submit worksheets at the specified level in the hierarchy.
- PRICE_USER – allows access to the UI.
- PRICE_VIEWER – has read-only access to worksheets at the specified level in the hierarchy.
- BRM_PRICE_EDIT – can edit a business rule through the UI at the item level or higher.
- BRM_PRICE_VIEW – can view a business rule through the UI at the item level or higher.
- BRM_PROFITLOGIC_EDIT – can edit administrative business rules.
- BRM_PROFITLOGIC_VIEW – can view administrative business rules.
- WHAT_IF_SERVICE_USER – similar to the PRICE_USER role, but for the web service.

Significant MDO actions are defined as follows.

- PRICE_COMMENTS_EDIT – can edit tool tips.
- PRICE_GUARD – guards against illegal access to the application.
- MDO_WHAT_IF_SERVICE_EXEC – provides access to the web service.
- PRICE_MAINTAINING_MERCHANDISE_VIEW_NO_PG_EDIT – provides read-only access to the Pricing Group Manager (only the action is available by default).
- PRICE_ITEM_INFO_VIEW – can view item information

- PRICE_SEASONALITY_VIEW – can view seasonality curves but cannot override or change mappings.
- PRICE_SEASONALITY_EDIT – can edit seasonality curves. This permission only applies to hierarchies that the user has permissions to edit.

Roles are assigned to users with restrictions that are defined at or above a specific node of the merchandise hierarchy and the location hierarchy. The scope of actions can be across the merchandise and location hierarchies. The scope must be defined at or above the class planning level.

The sample file, "Role Assignment Sample xml File" provides an illustration of defining the scope.

About User Management Roles

User accounts with user management roles have access to features such as creating users, assigning roles, removing user accounts, resetting passwords.

When a user with a user management role logs on, a link to the User Management utility appears on the Main Menu.

The following list describes the default User Management roles:

- UM_READ_ONLY_ADMIN – This role allows read-only access to the User Management utility. This role has privileges to view the list of users and their roles and hierarchy levels, but not to create new user accounts or modify or inactivate existing ones.
- UM_ROLE_ASSIGN_ADMIN – This role allows assigning new roles (and related hierarchy levels) to existing user accounts, but it does not allow the creation of new user accounts.
- UM_USER_ADMIN – This role allows creating new user accounts, but it does not allow the assignment of roles to the new accounts.

Password Policies

Here is a list of standard rules for a common password policy that you can use to customize useraccount.properties.

- Establish minimum and maximum password length requirements.
- Do not allow users to re-use the last *N* passwords during the password change procedure.
- Lock the account after *N* failed login attempts.
- Set an expiration length of time for the password.
- Prompt the user to change the password before expiration when the user logs in.
- Specify the length of time that a password is valid before expiration.
- If the password has expired, prompt the user to change the password when the user next logs in.
- Disable user accounts after *N* consecutive failed login attempts.

9

Model Run

This chapter contains:

Model Run Process

The weekly batch process involves entire process of preparing for an optimization run, including loading data and customizing Markdown Optimization parameters, performing the pre-optimization run steps, and performing the post-optimization run steps.

During an model run, Markdown Optimization analyzes business data and produces markdown recommendations and forecasts.

This chapter provides an overview of the entire model run process and details the steps of the optimization run process. See the MDO *Operations Guide* for technical configuration details.

For information about troubleshooting the model run, see the MDO *Operations Guide*.

Calc Engine Configuration

The Calc Engine is the software that computes markdown optimizations and forecasts for MDO.

The following two files, <ConfigRoot>/Engine/delphi.properties and <Configroot/Price/kpi.properties, should be configured. Note that the properties for delphi are loaded from files named delphi.properties, which are found by searching the following directories under configroot (in order): Engine/client; Engine/environment; and Engine/. If a file named delphi.properties is found, it is loaded. This overwrites any delphi.properties files that were previously loaded.

Settings for kpi.properties

- Database credentials
- **chunk.tryLimit** – defines the maximum number of times that the engine tries to process an item before deciding that the item cannot be processed. This value must be set to a value greater than 1. The default reflects the optimal policy according to simulations.
- **chunk.sizes** – defines a sequence of values representing the sequence of chunk sizes that should be used to group items that have had 0, 1, 2... retries. The sequence must

- consist of a decreasing set of positive integers
- equal in length the value of chunk.tryLimit
- end with a value of 1
- use semicolons to separate the series of values

The sequencing is used in a progressive manner, starting with a large chunk and retrying with smaller chunks, to determine which items are causing the chunk to fail. (Retries are seldom needed; they happen when processes die.)

The processing of large chunks takes up much of an optimization run, so the size of largest chunk has an impact on performance. The default sequence of values is recommended for good performance.

When selecting the largest chunk size, consider the following. Since larger chunks require more random access memory per worker, at some point processes will either fail or use virtual memory paging. Smaller chunks are better because they incur less overhead in database access. It is suggested that the largest chunk size be a value between 1,000 and 10,000. The smaller chunk sizes should be much smaller than the largest value in order to capture work before a failure occurs.

Default value = 10,000; 100; 1

- **worker.lifetime** – defines how long in minutes the processor is allowed to run before it is decided that it is in an infinite loop and terminates. This value must be greater than 1 minute. For large collections, it is recommended that you start with a setting greater than 30 minutes.

  Default value = 30

- **chunk.active** – defines the maximum time in minutes after a worker is killed that the chunk it was working on can be reclaimed. This value must be greater than 1 minute. This setting rarely needs customizing.

  Default value = 3

Settings for delphi.properties

The delphi.properties file should only list values that differ from the default values. If a default value exists, it is listed here. The first two properties, for the Agorai library location and the RMI server port, are required. All others are optional. Refer tot he MDO *Operations Guide* for a list of all the properties in delphi.properties

Model Run Prerequisites

Once the application is installed, the following must be configured prior to an model run.

- Merge any existing customized load statements with the updated **load_statements.sql**. The load statements are used for eligibility filtering and for populating the ITEM_DATA table. **Load_statements.sql** is installed under **db.config**.
- Merge any existing customized inference rules (located in **ir.sql**, installed under **db.config**) with the updated inference rules.
- Apply **load_statements.sql** and **ir.sql** to the database schema, using either **configdb.sh** or **plconfiguredb.sh**, as follows:
  - copy **ir.sql** and **load_statement.sql** to **$CONFIGROOT/db.config**

  - **cd $PL_BASE/modules/tools/bin**
    where **PL_BASE** is the location where Markdown Optimization is installed.
  - **bash plconfiguredb.sh $CONFIGROOT**
- Business Rule Definitions – Markdown Optimization comes with default values for business rules, set at the highest level (except Outdates and planned Start Dates). Merge any customized business rule definitions with the updated version.
- Business Rule Values – load using bulkloader.
- Set the LD_LIBRARY_PATH (for Linux, HP-UX, or Solaris) or LIBPATH (for AIX) as follows:

Platform	Pathname
AIX	LIBPATH=<install_dir>/modules/Engine/lib/aix_powerpc
Linux	LD_LIBRARY_PATH=<install_dir>/modules/Engine/lib/linux_i686
Solaris	LD_LIBRARY_PATH=<install_dir>/modules/Engine/lib/sunos_sun4u
HP-UX	LD_LIBRARY_PATH=<install_dir>/modules/Engine/lib/hpux_ia64

No special maintenance of tables or indexes is required. All necessary statistics gathering and index rebuilding is handled by the application.

The performance of the optimization run is affected by the number of candidate prices from the price ladder (those lower than the current price) and candidate markdown calendar (which depends on how far out the outdate is). Performance should be considered when choosing prices for the price ladder; if the price ladder is long, performance may be adversely impacted.

Model Run Process

The weekly Model Run consists of the following high-level steps. These steps are all executed by **weeklyBatch.sh**. (The details about each step are provided in subsequent sections.)

Note that since the model run and KPI share work_queue_tbl, you should not run KPIs and the model run at the same time.

1. **plfrontendload.sh**, which executes FELOAD
2. **plpremodelrun.sh**, which executes PRERUN
3. **runCalcEngine.sh**, which executes a series of helper scripts responsible for the batch process
4. **runMultiKPI(Item | Collection).sh**, which calculates the key performance indicator metrics
5. **plpostmodelrun.sh**, which executes POSTRUN
6. **refreshSummaryCache.sh**, which refreshes the P4P_WORKSHEET_SUMMARIES cache table
7. **refreshForecastCache.sh**, which refreshes the forecast cache

For a complete list of the scripts used during the model run, refer to the MDO *Operations Guide*.

Load_statements.sql

The following are contained in **load_statements.sql** and are part of the optimization run.

FELOAD

Prior to FELOAD, certain tables are cleaned up in order to improve performance. During the FELOAD portion of the optimization run, the ITEM_DATA table is updated with new data that has been received from the client.

Note that the Mhrename standard interface does not remove inactive cluster sets. So, items that are members of inactive cluster sets are filtered out during the population of the INTERNAL_ITEM_DATA_TBL table.

FELOAD consists of the following steps:

1. Procedures for archiving ITEM_DATA and Forecasts are loaded.
2. The SCENARIOS_TBL table is loaded.
3. The INTERNAL_PROMO table is loaded.
4. The INTERNAL_HIST_MKDNS_TBL table is loaded.
5. The INTERNAL_BIZ_RULES_TBL table is loaded.
6. The IR_WAREHOUSE_ALL_ITEMS_TBL table is loaded.
7. The ITEMS_TBL.model_start_dt is updated.
8. All invalid views are recompiled.
9. The INTERNAL_ITEM_DATA_TBL table is truncated.
10. The INTERNAL_ITEM_DATA_TBL table is populated.
11. All dropped indexes are restored on the internal ITEM_DATA table.
12. Statistics are collected on the internal ITEM_DATA table.
13. The FEDATES tag, which includes updating P4P_PARAMS, is created.
14. The COLLECTIONS tag, which truncates the P4P_COLLECTION table, is created. This step is not called by any other step in **load_statements.sql**.
15. The P4P_COLLECTION table is truncated.

Data Archiving Data in Markdown Optimization is archived in such a manner that data not required for the weekly optimization run or by the application is cleaned up regularly. Required data is preserved and performance is enhanced.

Two procedures, which are part of the FELOAD portion of **load_statements.sql**, are responsible for archiving:

- **com.profitlogic.db.birch.ArchiveForecasts**
- **com.profitlogic.db.birch.ArchiveItemData**

Archiving occurs once a week. The current data in each table is compared to the data in the archive and only the new records are selected to be appended to the archive.

The following table lists the archiving source tables and target tables.

Table 9–1 Archived Tables

Archive Source Table	Archive Target Table	Populated By	Amount of History Preserved
FORECAST_ACTIVITIES	FORECAST_ACTIVITIES_ARCH	Model Run	unlimited
FORECAST_RUNS	FORECAST_RUNS_ARCH	Model Run	unlimited
FORECAST_SUMMARIES	FORECAST_SUMMARIES_ARCH	Model Run	unlimited
MARKDOWN_ACTIVITIES	MARKDOWN_ACTIVITIES_ARCH	Model Run	unlimited
RTM_HISTORY	RTM_HISTORY_ARCH	Model Run	unlimited
RTM_STATUS	RTM_STATUS_ARCH	Model Run	unlimited
RUN_HISTORY	RUN_HISTORY_ARCH	Model Run	unlimited
ITEM_DATA	ITEM_DATA_ARCH	FELOAD, KPIs, user actions	two years

Forecast Archiving The forecast archiving process maintains a status table that is truncated at the beginning of each weekly archiving. As each step in the process is successfully completed, its completion status is logged into the status table (ARCHIVING_STATUS_TBL). The cleanup of the source database tables, which is the last step of the procedure, begins only after all required archiving steps have been completed.

ITEM_DATA Forecasting The ITEM_DATA forecasting step preserves all the columns from the ITEM_DATA table. The ITEM_DATA_ARCH table contains one additional column, CURRENT_WEEK, which is populated from the PARAM_NAME column (= CurrentWeek) in the P4P_PARAMS table. The ITEM_DATA archiving step included the truncating of the ITEM_DATA table.

PRERUN

During the PRERUN portion of the optimization run, the BRM business rules are cached into ITEM_BRM_RULES at the item level. Each column in the table represents a separate business rule (in correspondence to BRM_RULE_DEFINITION TBL). Each row in the table represents a unique ITEM_ID, with its business rules exploded to the item level. The ITEMS view along with the SEASONALITY_ID are cached in ITEMS_MODELRUN_TBL, which is used as a source for most inference rules.

Note that chunking has been disabled by default in the BRM caching step. Chunking should only be enabled if processing is too slow. To enable chunking, configure the BRMChunkSize parameter in P4P_PARAM as follows:

Table 9–2 BRMChunkSize Parameter Values

BRMChunkSize Value	Description
0	No chunking (the default)
> 0	Chunks created of size specified
< 0	Chunking not allowed

The value can be set manually or by using the script **..\modules\tools\bin\plsetbrmchunksize.sh**. The value is set to the default value of 0 at the time of the initial load.

PRERUN consists of the following steps:

1. Tag added to disable Model Run Indexes.
2. The BRM business rules are cached into the ITEM_BRM_RULES table.
3. All indexes are dropped on the ITEM_MODELRUN_TBL table.
4. The ITEM_MODELRUN_TBL table is truncated.
5. All What If output tables are truncated.
6. The WIF_SCENARIO_TBL table is populated with the required row of seed data.
7. FULL_PRICE and PRICE_LADDERS are cached for items.
8. The ITEM_MODELRUN_TBL table is populated.
9. All dropped indexes are restored on the ITEM_MODELRUN_TBL table.
10. The IR_WAREHOUSE_CACHE_TBL, which is a cache for IR_WAREHOUSE, is loaded.
11. BRM_ATTRIBUTE_VALUE_TBL is populated.
12. HIST_MARKDOWNS_MODELRUN_TBL (HIST_MARKDOWNS cache) is populated.
13. The PERIODS_MD_CAL_TBL table, which is a subset of PERIOD_TBL, used by the IR_MARKDOWN_CALENDAR view, is populated.
14. The P4P_LADDER_ROLES_TBL is populated.
15. All table statistics are updated.
16. All invalid views and schemas are recompiled.
17. The Model Run indexes and constraints are disabled.
18. The IR_METRICS cache tables are populated.

POSTRUN

During the POSTRUN portion of the model run, the P4P_FORECAST_DATA table is updated with the latest model run results, the ITEM_DATA table is updated with certain metrics, and RECOMMENDED_COLLECTION_FLAG and WORKSHEET_ID in P4P_COLLECTIONS are updated with data from the ITEM_DATA table.

POSTRUN consists of the following steps:

1. All indexes are dropped on the P4P_FORECAST_DATA table.
2. The P4P_FORECAST_DATA and TMP_P4P_FORECAST_DATA tables are truncated.
3. The TMP_P4P_FORECAST_DATA table is loaded.
4. Statistics are collected on the P4P_FORECAST_DATA table.
5. The P4P_FORECAST_DATA table is loaded.
6. All dropped indexes are restored on the P4P_FORECAST_DATA table by the RestoreTable procedure.
7. Statistics are collected on the P4P_FORECAST_DATA table.

8. The TMP_P4P_FORECAST_DATA table is truncated in preparation for the load.
9. All indexes are dropped on TMP_POSTRUN_ITEM_DATA.
10. The TMP_POSTRUN_ITEM_DATA table is truncated.
11. The temp table used in the ITEM_DATA update to populate proj_oh_units_eff_dt is loaded.
12. All dropped indexes on TMP_POSTRUN_ITEM_DATA are restored using the RestoreTable procedure.
13. Statistics are collected on the TMP_POSTRUN_ITEM_DATA table.
14. The COLLECTION_NAME, PROJ_OH_UNITS_EFF_DT, and RECOMMENDED_COLLECTION_FLAG columns in ITEM_DATA are updated, based on the most recent model run results.
15. Statistics are collected on the ITEM_DATA table.
16. The P4P_COLLECTION.RECOMMENDED_COLLECTION_FLAG column is updated with the latest information from ITEM_DATA.
17. Statistics are collected on the P4P_COLLECTION table.
18. All invalid schema objects are recompiled.
19. All invalid views are recompiled.

Summary Metrics

The P4P_WORKSHEET_SUMMARIES table stores the aggregate data for all summary metrics for all worksheets. The P4P_WORKSHEET_SUMMARIES table is indexed after the data is populated, and the name of the index is SummaryCache_IDX.

This table is initially created with one column, the worksheet_ID column, when the application is deployed. The Worksheet Summaries page in the application obtains summary metrics from P4P_WORKSHEET_SUMMARIES. Parameters associated with the worksheet summary cache are stored in P4P_SUMMARYCACHE_PARAMS. The parameters table should always exist. The summary metrics table can be rebuilt or refreshed.

When P4P_WORKSHEET_SUMMARIES table is rebuilt, the existing table is dropped, then recreated and populated with data. When the table is refreshed, all rows in the table are deleted and re-populated.

A refresh of the cache table is triggered when

- the application server is running and the **refreshSummaryCache.sh** command is invoked

The cached table is rebuilt when:

- the application server is running and the configuration has changed and the **refreshSummaryCache.sh** command is invoked.
- the configuration has changed and the application server is restarted.

If the application server is restarted but the configuration has not changed, then the cache table is not rebuilt or refreshed. So, P4P_WORKSHEET_SUMMARIES should be refreshed/rebuilt after every model run and whenever changes are made to xml files in the p4pgui/grids directory, or if a change is made from the UI that affects the summary metrics, that is, taking a markdown via a worksheet.

In addition, grids are refreshed through **p4padmin.jsp** when the application server is running. Clicking on the Grid Configuration link reloads the grids and rebuilds the summary metrics cache.

A link, Worksheet Summary Cache Information, provides the following diagnostic information:

- the names of the cached columns
- the worksheet IDs that are cached
- the SQL statement used to calculate the summary metrics

Monitoring an Optimization Run

The commands you can use to monitor the progress of the optimization run include:

- **getCurrentJob.sh**
- **getCurrentJobStatus.sh**
- **isDone.sh**
- **jobHistory.sh**
- **jobReport.sh**
- **runReport.sh**

The optimization run can be monitored by reviewing the exit codes for the worker processes from **runCalcEngine.sh** and **multiChunker.sh**.

Resource monitoring of the application host that the worker processes are running on and the database host that the worker processes communicate with is also recommended. The saturation or overuse of hardware can indicate a configuration problem, such as the wrong number of worker processes, the wrong number of worker processes per machine, the wrong chunk size, or inappropriate heartbeat times.

The **isDone.sh** utility returns an exit code of 0 if the current batch run is complete; otherwise, it returns an exit code of 1.

The **getCurrentJobStatus.sh** utility prints a number between 0 and 100, which represents the approximate percentage of processing completed. This value is computed as a weighted percentage of completed chunks from the work queue, so the value is less accurate if business rules are more heterogeneous across merchandise or if the chunks are large.

The **jobReport.sh** utility prints a detailed breakdown of the number of items completed, the number of collections completed, and the number of optimizations of each type that have failed.

The **runReport.sh** utility prints the Stoplight Summary. This is available at any time during the batch process, but it does not indicate if the job is complete.

These monitoring scripts provide a database-level view of how the run is proceeding. However, monitoring the exit status of the worker processes for unexpected failures is also recommended. These unexpected failures may indicate a configuration or data problem such as overly aggressive suicide times or problems with inference rule customization. It is also recommended that you redirect ***stderr*** to a log file in order to view any warning messages.

Sendback Files

A sendback file is a file that is created using the Markdown Optimization Admin command, **generateSendback**. The sendback file contains information about changes made via the application user interface. The file is the mechanism for transmitting the updated markdown information to the client. The content of the sendback file is determined by the SQL query.

Usually, a sendback file is generated once a week at the Sendback Date, which occurs at the Cutoff Date and Time. However, the schedule for generating and transmitting a sendback file is determined by the needs of the business. Individual retailers may require sendback files at regular intervals during the week or may even require daily sendback files. The script to automate the sendback process should be designed to manage the schedule.

Sendback File Example

The following example shows the generation of a sendback file with a typical weekly schedule, called the markdown cycle:

Only one sendback file is included in this example; however, retailers may require the generation of more than one sendback file as part of their standard schedule.

To generate a sendback file, you must:

- write one or more SQL queries against the database. A query should specify the name (type) of the sendback file, line ending, file delimiter, and information to be included in the sendback file, such as item, location, value of the new price after markdown, and date of price change. Edit the **p4pgui-config.xml** file in the configuration root directory to include the necessary queries

  The following sample query generates information about outdates:

  ```
  <sendback-query name="pl-outdates"
  line-endings="unix" result-delimiter="|"><![CDATA
  [select item_id, modified_out_of_stock_date from
  p4p_items where modified_out_of_stock_date is not
  null]]></sendback-query>
  ```

- edit the property, **p4pgui.sendback.dir = pathname** in the **configuration_root/p4pgui/config.properties** file to specify the destination of the generated sendback file.
- create a script to automate the scheduled weekly generation of the sendback file that includes the necessary Markdown Optimization Admin commands.

Key Performance Indicators

The model run calculates Key Performance Indicators (KPIs) metrics. Some of these metrics depend on the model run and some do not. Inference rules provide a configurable interface for all KPI calculations. Many inference rules are used by the KPIs, but only a few of them are configured frequently. The KPI step of the model run updates records in the ITEM_DATA_TBL separately. The inference rules that are configured for the KPIs include:

KPIs	Inference Rules
Defines the forercastable inventory was included in the optimization.	IR_HISTORIC_METRICS

KPIs	Inference Rules
Defines the custom metrics that do not depend on the forecast.	IR_USER_FLOATS IR_USER_DATES IR_USER_TEXTS
Defines the custom metrics that depend on the forecast.	IR_O_USER_FLOATS IR_O_USER_DATES IR_O_USER_TEXTS
Defines the cumulative sell-through %, which is a configured calculation.	IR_ROLLUPS

Notes for Model Run Configuration Points

The model run is the MDO weekly batch process that uses the forecasting model produced by APC to perform the optimization using the client-specific business rules. The output of the model are recommendations regarding when and how deeply to discount merchandise in order to achieve the highest gross margin dollars over the entire life cycle to the defined out date. Key performance indicators, which are metrics such as Gross Margin, are also produced. These metrics are used to populate the MDO UI display.

In addition to business rules, the following must be configured:

Level of Optimization (Merchandise and Location Hierarchy) – the level of optimization determines the level at which the model run makes markdown recommendations. Considerations include: for the merchandise hierarchy, the style and the color (will markdowns be taken differently by color). For the location hierarchy, the Chain and the Cluster (will markdowns be taken differently for clusters?). Items are defined as the intersection of a node in the MH and a node in the LH. The level used in each hierarchy is a configuration choice, but must be the same for all items. For example, items might be defined at the Color-Region level. MDO aggregates sales data to the Item level and persists sales data at the Item level only. MDO optimization occurs at the Item level and/or as a pricing group of Items. Defining the optimization level is a critical configuration decision. See details regarding ASH_CP_TBL, LH Levels, and MH Levels.

Markdown Worksheet Level – when configuring the Item Worksheet level, take into consideration that one user can be in a worksheet at a time. Pricing Groups cannot span worksheets. Worksheets are typically tied to user workflow and area of responsibility. Budget is fed at the worksheet level (OTB). All of items that are grouped together for management in the UI are typically assigned to a single MDO user. For example, a worksheet may contain all the items in a department.

Application Availability – when configuring this, take into considerations that this defines the application up-time during the markdown week. It is dictated by contractual service level agreements (SLAs) and business process requirements. Data receipt timing and volume are key considerations. Backup and scheduled maintenance must also be considered.

Markdown Calendar / Forecast Implications – defines the weeks in which markdowns are allowed. In cases where the markdown calendar is sparse, configuration decisions must be made concerning whether to display recommendations in weeks during which an action can be taken (dark weeks). Display recommendations each week forecasted for the next markdown eligible week.

Markdown Day of the Week – defines the calendar day of week on which markdowns are effective in stores. Are markdowns effective on the same day for all departments?

Markdown Cutoff Timing – defines the day and time by which markdowns must be approved in the applications. This is determined by business process, store execution, and price change system constraints and is displayed in the UI.

Markdown Sendback Timing and Format – defines the day or days and time when a sendback file is created with approved price changes. The format must support integration with the price change execution system or keying report. See the Standard Load chapter for script.

Business Rule Maintenance Cutoff – defines day and time by which changes to pricing groups, exit dates, and inventory sell thru targets must be completed in the application. This must be completed prior to the next model run.

Forecastable Inventory – defines the inventory that should be considered by the model for optimization. This typically includes Store On hand, Store On order, DC On hand, and DC on order. All forecastable inventory is typically included in markdown accounting metrics.

Virtual Allocation – the allocation of unallocated inventory. This defines the assumed distribution of central, uncommitted inventory to regions, clusters, or stores. It is used for below CHAIN level optimization. It does not necessarily relate to distribution reality.

Item Eligibility – defines the criteria that must be met for an item to get loaded into the front-end application.

10

Localization

This chapter contains the following:

Introduction

Internationalization is the process of creating software that can be easily translated. Changes to the code are not specific to any particular market. Markdown Optimization has been internationalized to support multiple languages.

This chapter describes configuration settings and features of the software that ensure that the base application can handle multiple languages.

Translation

Translation is the process of interpreting and adapting text from one language into another. Although the code itself is not translated, components of the application that are translated include:

- Graphical user interface (GUI)
- Error messages

The following components are not translated:

- Documentation (Online Help, Release Notes, Installation Guide, User Guide, Operations Guide)
- Batch programs and messages
- Log files
- Configuration Tools
- Reports
- Demo data
- Training materials

The user interface for Markdown Optimization has been translated into:

- Chinese (Traditional)
- Chinese (Simplified)
- French

- German
- Italian
- Japanese
- Korean
- Portuguese (Brazilian)
- Russian
- Spanish (Spain)

Markdown Optimization depends on both the browser settings and the regional settings to determine which language is being supported for a specific implementation.

Markdown Optimization does support multiple languages within a single installation. It does not support multiple currencies within a single installation.

11

Production

This chapter contains:

Introduction

This chapter provides an overview of implementing a production process.

For more information, see the *MDO Configuration Guide.*

Process

Here is a high-level overview of the production process.

- Set up a production environment. Refer to the MDO *Installation Guide* for details about the hardware, networking, and software requirements and relevant procedures.
- Set up the production ftp connection for the transfer of the weekly data feeds and MDO export files.
- Migrate the test database to the production database server.
- Deploy the customized MDO configuration in the production environment.
- Set up automation.
- Set up a scheduler.
- Set up a monitoring system to capture batch process failures to the error log file for troubleshooting purposes.
- Monitor performance in order to determine whether database performance should be adjusted.
- Establish a database backup process.
- Verify that the SLA is being met.
- Assign access privileges.
- Define file exports.

Weekly Operational Schedule

The following events comprise the weekly MDO schedule. The timing of these events is client specific.

- Application Availability – This is affected by when the weekly client data is received, the schedule of data backups and system maintenance, and the weekend batch process schedule.
- Markdown Cutoff Timing – During the week, certain activities can only be performed at certain times. The markdown decision period is usually scheduled between Monday and Thursday. This ends at the time indicated by the cutoff date displayed in the UI. The cutoff date defines the day and time by which markdowns must be approved in the application. The cutoff date defines the date and time by which changes to the BRM must be completed in the application. It defines the day and time by which changes to pricing groups, exit dates, and inventory sell through target must be completed in the application. This date is determined by business processes, store execution, and price change system constraints.
- Read-only Period – This period occurs after the cutoff date. During this time, the submitted markdowns and other data are entered into the system.
- Model Run – the weekly batch process occurs every weekend. During this time, forecasts and markdown recommendations are generated and the application is not available.

Automation

Automation is generally used during the weekly process to manage the entire schedule of activities. All weekly MDO processes are typically automated including data delivery, staging and loading of data, running the model run, and generating the sendback files. The custom scripts that are used are developed during the implementation process.

The weekly process consists of the following steps.

Loading Weekly Data

This step loads the client data into the MDO schema.

Staging Weekly Data

This step is performed using the pl_stage_Client.sh script. This script is typically customized to reflect the client's directory structure.

Loading Staged Data

The staged data is loaded into the MDO tables using the pl_load_client.sh. This master script contains load procedures for all the data feeds, including those that are not required on a weekly basis, such as demand parameters, and data feeds that are never used by a specific client. These scripts can be removed.

Model Run

The model run executes a series of scripts. The details about these scripts can be found in the Model Run chapter. Custom procedures are generally added to the Post Run step.

On Demand

Oracle provides a hosting option for clients who want a turnkey solution for their implementation. The Oracle On Demand hosting solution provides:

- a proven architecture and network model
- Oracle-provided services
- new, dedicated hardware
- Oracle support backed by vendors and partners
- secure facilities

12

Best Practices

This chapter contains:

Introduction

This chapter describes best practices for the MDO implementation.

For more information, see the MDO *Operations Guide* and *Configuration Guide*.

Implementation Best Practices

When implementing MDO, keep the following best practices in mind.

Loading Sales History

- Request weekly files from the client.
- Stage and load one week of data at a time, especially the initial data used for testing.
- Set the threshold values for error handling high (approximately 25%) when loading the history files. This value is set in the properties file, **dbError.properties**.
- If you create all possible item (Merchandise Hierarchy/Location Hierarchy) combinations when loading the historical data, you should delete these items before Go Live.

File Naming Conventions

Use the following conventions for naming files:

File Type	Example
Grids	mm-description.xml
Reports	rpt-description.xml
Database objects	ISC_object (Implementation Services Customization
Temporary database objects that are not required for production	X_ISC

File Type	Example
SQL file	Named the same as the object (e.g., p4p_ladder_roles.sql)

Documenting Code

When you change a configuration, such as an IR view in custom_ir.sql, you should enter a comment before the view definition that explains the reason for the configuration change and that references the requirement ID in the solution design document.

You should use meaningful comments when you check code into the source control system. Never check out all the files as once.

Product Files

Do not edit the following files:

- p4p-column-list.xml. Instead, use p4p-custom-columns.xml. Only new or modified metrics go into the custom file.
- loadstatements.sql. Instead, use custom_load_statements.sql. Only modified statements go into the custom file.
- ir.sql. Instead, use custom_ir.sql. Only modified views go into the custom file. Put views directly related to the inference rules into this file. (For example, do not add sendback views into this file.)
- onetime_db_updates.txt should contain any custom tables or indexes that you have added as part of the implementation.
- regular_db_updates.txt should contain any custom views, functions, procedures, and packages that have been created or customized. The file should also contain product_ir.sql and loadstatements.sql before custom_ir.sql and custom_ loadstatments.sql.

Eligibility and Load Statements

- Add as little as possible into loadstatements.sql.
- Use ISC_IR_ELIGIBILITY and ISC_IR_HIERARCHY.

Prerun and Postrun

- For ease of testing and modification, use the package to store all PRERUN and POSTRUN code instead of adding the SQL to loadstatements.sql.
- Packages should be named ISC_PRE_RUN and ISC_POST_RUN.
- Add calls to the automation list instead the loadstatements.sql file.

Pricing Groups

- If auto-collections are not used, you should add 1=0 to IR_ITEM_COLLECTION, in case the view has not been removed from the automation list.
- You should review all IR_XXX_C views.

- Update O_USER_FIELDS in the postrun. Both IR_O_USER_XXX and IR_O_USER_XXX_C are used to update the O_USER fields. If the views return different results, unexpected behavior can result.

External Markdown Sendback

- The default sendback file matches the MDTaken inbound file.
- Use the view ISC_EXT_MKDN_SENDBACK.
- All external sendbacks should be defined using the database view.
- You should not create a complex SQL in the p4pgui-config.xml file. Instead, create a sendback element that selects from the view.
- The pre-sendback-update tag should use a function to update the ITEM_DATA table.

  However, if OAS is used as the application server, a function call will not be accepted. Instead, you should use a SQL block to call the function.

  For example, DECLARE r number; BEGIBN r:=ISC_EXT_MKDN_SENDBACK_UPDATE;END;
- All pipe delimiting should be done by the product. It should not be done in the external sendback view.

Other

- If internal sendbacks are used, add 1=0 to PL_MARKDOWN_SENDBACK in case the view is not removed from the automation list.
- All DB scripts should be checked into the source control system. This includes custom scripts used to upgrade or migrate a client.
- One-time data loads in either XLS or CSV format should be checked into source control.