Oracle® Demantra

Analytical Engine Guide

Release 12.2

Part No. E44444-05

August 2016

ORACLE

Oracle Demantra Analytical Engine Guide, Release 12.2

Part No. E44444-05

Primary Author: Greg Watkins

Contents

8 Using the Engine Administrator and Running the Engine

9 Engine Details

10 Engine Parameters

11 Theoretical Engine Models

Index

Send Us Your Comments

Oracle Demantra Analytical Engine Guide, Release 12.2

Part No. E44444-05

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

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

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

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Oracle E-Business Suite Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

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

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

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

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

Preface

Intended Audience

Welcome to Release 12.2 of the *Oracle Demantra Analytical Engine Guide.*

See Related Information Sources on page xi for more Oracle E-Business Suite product information.

Documentation Accessibility

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

Access to Oracle Support

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

Structure

1 Introduction to the Analytical Engine

> **Important:** The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes the role the Analytical Engine plays in the simulating forecasts. It also describes how the Analytical Engine functions generally and the different engine modes.

2 Basic Concepts

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter introduces the basic concepts involved with configuring the Analytical Engine.

3 Configuring the Analytical Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to configure the Analytical Engine. It also introduces guidelines for configuring the forecast tree, causal factors, and the configure to order (CTO) feature.

4 Configuring the Forecast Tree

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to configure the forecast tree. In the case of PE mode, it also describes how to configure the influence relationships, and competition among the combinations.

5 Configuring Causal Factors

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to create causal factors, configure them, and populate them with data. It also describes the predefined causal factors provided by Demantra.

6 Configuring Promotions and Promotional Causal Factors

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to configure promotions and promotional causal factors in the Business Modeler.

7 Tuning the Analytical Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

It is usually necessary to adjust some parameters to configure the Analytical Engine correctly before running it the first time. Other adjustments can be made later to optimize the behavior and performance.

8 Using the Engine Administrator and Running the Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

Before you run the Analytical Engine for the first time, it is useful to ensure that you have configured it correctly. This chapter describes how to administer the Analytical Engine.

9 Engine Details

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter provides details on the Analytical Engine, for the benefit of advanced users.

10 Engine Parameters

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes the Analytical Engine parameters that you can see in Business Modeler and lists their default values, if any.

11 Theoretical Engine Models

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter contains reference information for the theoretical models that the Analytical Engine uses.

Related Information Sources

Oracle Demantra products share business and setup information with other Oracle Applications products. Therefore, refer to other user guides when you set up and use Oracle Demantra.

User Guides Related to All Products:

- *Oracle Applications User Guide*
- *Oracle Applications Developer's Guide*
- *Oracle E-Business Suite Concepts*

User Guides Related to Oracle Demantra:

- *Oracle Demantra User's Guide*

- *Oracle Demantra Installation Guide*
- *Oracle Demantra Implementation Guide*
- *Oracle Demantra Demand Management User's Guide*
- *Oracle Demantra Deduction and Settlement Management User's Guide*
- *Oracle Demantra Trade Promotion Planning User's Guide*

Integration Repository

The Oracle Integration Repository is a compilation of information about the service endpoints exposed by the Oracle E-Business Suite of applications. It provides a complete catalog of Oracle E-Business Suite's business service interfaces. The tool lets users easily discover and deploy the appropriate business service interface for integration with any system, application, or business partner.

The Oracle Integration Repository is shipped as part of the Oracle E-Business Suite. As your instance is patched, the repository is automatically updated with content appropriate for the precise revisions of interfaces in your environment.

Do Not Use Database Tools to Modify Oracle E-Business Suite Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle E-Business Suite data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle E-Business Suite data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle E-Business Suite tables are interrelated, any change you make using an Oracle E-Business Suite form can update many tables at once. But when you modify Oracle E-Business Suite data using anything other than Oracle E-Business Suite, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle E-Business Suite.

When you use Oracle E-Business Suite to modify your data, Oracle E-Business Suite automatically checks that your changes are valid. Oracle E-Business Suite also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

1

Introduction to the Analytical Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes the role the Analytical Engine plays in the simulating forecasts. It also describes how the Analytical Engine functions generally and the different engine modes.

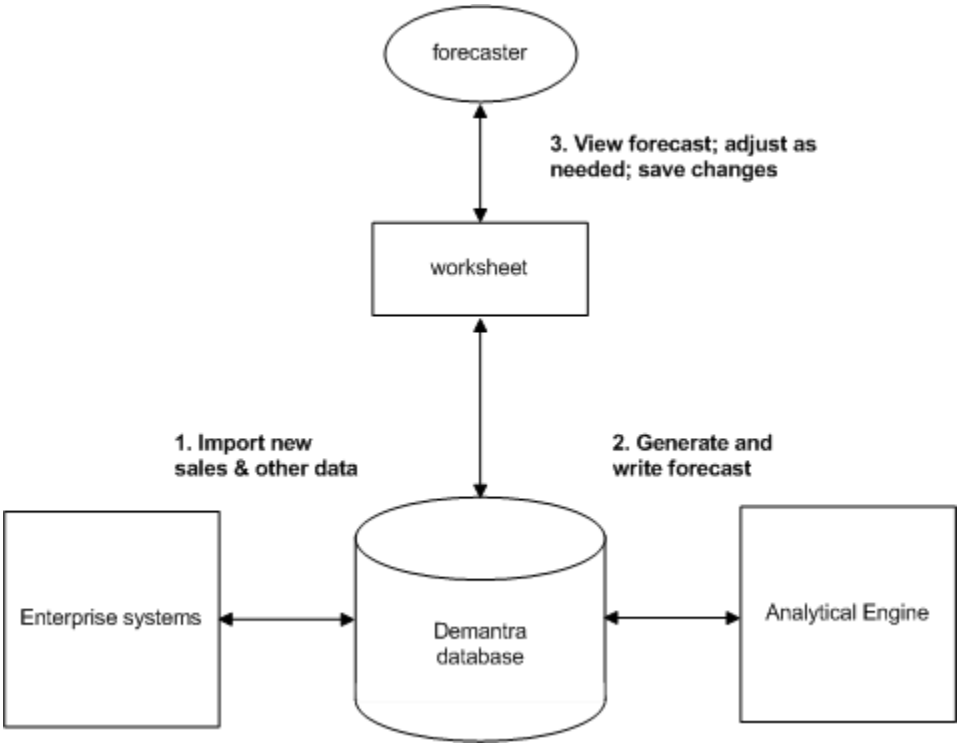
This chapter covers the following topics:

- Overview
- Engine Modes: DP and PE
- What the Engine Does
- Forecast Modes
- Engine Profiles
- Specifying the Demand Stream in an Engine Profile
- Maintaining Engine Versions
- Illegal Characters in Demantra

Overview

The Oracle Analytical Engine is an advanced Analytical Engine capable of multidimensional forecasting with mixed modeling techniques. The system is designed for large-scale installations handling analysis and modeling of tens to hundreds of thousands of different demand patterns.

The following figure shows an overview of how a Demantra solution uses the Analytical Engine:



Within a Demantra solution, the Analytical Engine runs periodically (in the background), reading data from the Demantra database and generating forecast data. The forecaster uses a worksheet to view the forecast and make adjustments, saving those changes to the database. The updated forecast is available to all users with the appropriate authorization.

The preceding figure is not meant to show hardware configuration, which is discussed in the Oracle Demantra Installation Guide, rather than in this manual. You should be aware, however, that the Analytical Engine can be used in a distributed mode. Specifically, your system may include the Distributed Engine, where the Analytical Engine is registered on multiple machines, all with access to the Demantra database. In this mode, the Analytical Engine automatically distributes its work across multiple machines simultaneously. This maximizes processing power and reduces bottlenecks. For more information about the distributed engine, see the Oracle Demantra Installation Guide.

Engine Modes: DP and PE

Oracle provides two different modes of the Analytical Engine:

- In PE mode, the engine is suitable for use with Promotion Effectiveness.
- In DP mode, the engine is suitable for use in demand planning applications.

What the Engine Does

The Analytical Engine accesses the database and reads the historical demand and data from the causal factors (such as seasons, price changes, and specific events such as promotions [in the case of Promotion Effectiveness). It then generates a forecast for all or specific item-location combinations. Wherever possible, it generates the forecast at the lowest possible allowed level (such as SKU-store). If necessary, it aggregates data so that it can generate a forecast at a higher level and split it to the lower level as needed. The forecast tree (which you configure) controls how the Analytical Engine aggregates and splits data when performing this task.

When working on a node of the forecast tree, the Analytical Engine uses a set of engine models, which are mathematical forecasting models. It considers how well each of those models works for that node and it statistically combines the best results, and generates the forecast from that. Advanced users may choose to adjust parameters that control how the individual models work; see "Theoretical Engine Models" for details on the models. Advanced users can also adjust how the Analytical Engine treats different nodes in the forecast tree.

In PE mode, the Analytical Engine also decomposes the forecast into the following:

- The baseline forecast (the forecast that would apply if no promotions were planned for the future)
- Direct effects (uplifts on item-location combinations due to promotions for those combinations).
- Switching effects (positive and negative effects on combinations due to promotions for other combinations)

See also

"Basic Concepts" "Engine Details"

Forecast Modes

The Analytical Engine can run in three modes: batch, simulation, or subset forecast.

Note: The Analytical Engine can run in only one mode at a time.

- In batch mode, the Analytical Engine considers all the item-location combinations and generates a forecast for all of them (with a few exceptions, noted in the next chapter). In a typical implementation, the engine automatically runs in batch mode regularly. Batch mode should be run separately of data load, typically after new data is imported.

- In simulation mode, the Analytical Engine considers only a subset of the combinations. In this mode, the engine (called the Simulation Engine) waits for simulation requests and then processes them.

  In simulation mode, a user runs a worksheet and submits a simulation request for some or all of the combinations in it. The simulation request is processed in the background but generally fairly soon. When the simulation is done, Demantra alerts the user, who can then accept or reject the results.

  In this mode, the user is usually performing a "what if" analysis, which refers to making some changes within the worksheet and then performing the simulation to see whether those changes have the desired effect. This process can be repeated until the optimum results are achieved.

  It is also possible to run simulations programmatically from within a workflow.

- In Subset Forecasting mode, the main differences are as follows:

  The column into which the engine will write into will be based on the Parent Batch profile associated with this profile; it will not receive its own set of columns and forecast versions.

  When engine is run using a subset type engine forecast, the existing (latest) forecast column for the parent profile will be used. No new engine version will be generated; when completed the engine will still appear the same column as latest version.

For more information about forecast modes, refer to Comparing Forecast Modes in this document.

"Running the Engine from the Start Menu"

Engine Profiles

Each engine profile is a set of parameters with specific values, causal factors, forecasting models, and a forecast tree. Engine profiles are available to quickly change the functioning of the Analytical Engine dependent on the type of forecasts you want to develop. Oracle Demantra provides some predefined profiles, and you can define additional engine profiles, as needed. When you run the Analytical Engine, you specify the engine profile to use.

The predefined profiles are as follows:

Base

This engine profile is the standard default Demantra engine.

Batch

The batch engine profile uses the same forecast tree and causal factors as the Base

engine, but the system parameters can be modified individually to reflect a different demand stream or other customization without modifying the Base.

Booking Forecast

The Booking Forecast engine profile uses the same forecast tree and causal factors as the default Demantra engine, but the system parameters can be modified individually to reflect a different demand stream or other customization without modifying the Base.

DSR POS Forecast

The DSR POS Forecast engine profile supports the Demand Signal Repository Point of Sale functionality. It uses the same forecast tree and causal factors as the default Demantra engine, but the system parameters can be modified individually to reflect a different demand stream or other customization without modifying the Base.

Forecast Install Base

This engine supports the forecasting of install base under contract, a service parts forecasting function.

The forecast tree is defined as follows:

Level	Item Level	Location Level
1	Lowest Item	Lowest Location
2	Item	Organization
3	Item	Organization Type
4	Highest Item	Highest Location

The causal factors associated with this engine profile are:

- Constant
- Trend
- Consensus Forecast

All other causal factors are disabled.

Forecast Spares Demand

This engine profile supports forecasting of spares at an organization. It executes on the

data and combination tables used by the Spares general level.

The forecast tree is defined as follows:

Level	Item Level	Location Level
1	Lowest Spares Level	Lowest Location
2	Latest Revision	Organization
3	Latest Revision	Organization Type
4	Highest Item	Highest Location

The causal factors are defined as follows:

- All existing defaults
- Install Base Under Contract

Forecast Non-Unit Maintenance Plan (UMP) Work Orders

This engine profile supports forecasting of work orders not associated with standard maintenance activity and service requests. The work order projection can be used as an input for processes outside this application generating future visits not linked to standard maintenance activity.

Forecast tree for new profile will be set as follows:

Level	Item Level	Location Level
1	Lowest Item Level	Lowest Location Level
2	Asset Group	Lowest Location Level
3	Highest Fictive Level	Highest Fictive Level

Engine Parameters

The following parameters differentiate this profile from other profiles:

- Min_fore_level=1
- Max_fore_level=2

- Parameter 'PopulationExtraFilter' will be configured to filter out only work orders associated with non-Unit Maintenance Plan (UMP) visits. The parameter should be set to a filter on the Visit Type level to only include the members which represent non-maintenance activity.

  Refer to Engine Parameters, page 10-1 for details about PopulationExtraFilter.

Simulation

This engine profile is the standard default simulation engine.

Simulation Install Base

The Simulation Install Base is a child of the Forecast Install Base engine profile. Should be used when generating a simulation on Install Base.

Simulation Spares Demand

The Simulation Spares Demand profile is a child of the Forecast Spares Demand engine profile. Should be used when generating a simulation on Spares Demand.

To create an engine profile:

When you create an engine profile, it is associated with a specific init_params table. It must not be the same table used by the other engine profiles. To check which init_params tables are in use, you can use the sql command `select * from engine_profiles;`

1. Navigate to System Parameters.

   Business Modeler > Parameters > System Parameters.

   The System Parameters window appears.

2. Select the Engine tab.

   The existing engine profiles are displayed in the Engine Profile drop-down menu.

3. Click New.

   The Create Engine Profile dialog box appears.

4. Select the engine profile you would like to use as a base for your new profile, if desired.

5. In the Profile Name field, enter the name of the engine profile.

6. In the Init Params Table Name, enter the init params table to be associated with this engine profile.

7. In the Profile Type field, select the appropriate profile type: Batch, Simulation Engine, or Subset Forecasting.

8. If you have selected Simulation Engine or Subset Forecasting as the Profile Type, use the Select Parent batch Profile list to assign the appropriate parent profile.

   Note: When choosing a Subset Forecasting profile it is recommended the same profile be chosen as profile it is based on and parent profile. If base and parent profiles are not the same, all parameters of newly created profile need to be reviewed to ensure settings are valid and match configuration of parent profile. For more information, refer to Subset Forecasting Mode Characteristics.

9. Click OK.

   The engine profile is saved.

Related Topics

Configuring the Analytical Engine, page 3-x

Engine Parameters, page 10-1

Specifying the Demand Stream in an Engine Profile

Statistical forecasts are calculated by analyzing a historical data stream to find trends and seasonal patterns in the data, and then projecting those trends and patterns into the future. One of the keys to this process is the historical data stream. This data stream is typically referred to as the *demand stream*. Often there are several options for which data stream to use, with typical examples being shipments, orders, and consumption data.

The decision on which demand stream to use is typically based on:

- Availability of data
- Quality of data
- Business problem being addressed

Sometimes, it is desirable to create forecasts based on historical data taken from more than one demand stream. Some sample scenarios are the following:

- A health food company wants to base its production forecast on orders. Orders are readily available for all stock keeping units (SKUs) and customers. This data serves as the basis for generating a forecast. For key customers, point of sales consumption data is available. This information gives the best insight on true weekly consumption and consumer behavior. For these customers, it is desired to generate

a forecast based on the point of sales data. This forecast can be compared with the order-based forecast as well as allow vendor managed inventory (VMI) relationships. Ad hoc what if analysis (simulation) is required for both data streams.

- A home goods company wishes to generate two forecasts. One forecast is based on shipments, while the second is based on consumer orders. The two forecasts are then compared. Areas with large differences are analyzed for forecasting anomalies. Order data is available weekly, while, due to data collection limitations, shipment information is only available every two weeks. This requires that the forecast generated based on shipments be run every 2 weeks while the order forecast is run weekly. Customer IT best practice requires the system to be able to display 5 versions of the order forecast and 8 versions of the shipment forecast.

Controlling which data series is used as the demand stream by the Oracle Demantra Analytical Engine is done by configuring the quantity_form system parameter. The quantity_form parameter contains an expression that is used by the engine to retrieve and aggregate demand stream data from the SALES_DATA table. Changing the expression results in different demand streams, or different combinations of demand streams, being used as the basis for generating the forecast.

To define the quantity_form parameter for an engine profile:

1. Navigate to System Parameters.

   Business Modeler > Parameters > System Parameters.

   The System Parameters window appears.

2. Select the Engine tab

3. Select the forecast from the Engine Profiles drop-down menu.

4. Navigate to the Data Manipulation sub tab.

5. Specify the expression in the Value column for the quantity_form profile row. The expression can be simple or complex. Make sure the quantity_form expression for a specific engine profile points to database columns containing the historical demand stream desired for the profile.

   The default syntax is: greatest(nvl(pseudo_sale,actual_quantity)*(1 + nvl(demand_fact,0)),0). This expression checks for the availability of user overrides stored in the pseudo_sale column. If they exist, these overrides supersede the historical sales found in the actual_quantity column. Any user defined % increase is then applied.

   Example

   - This is an example of a simple expression:

actual_quantity

> **Note:** When running the Analytical Engine the expression will be wrapped by a 'sum' function, therefore aggregation functions should not be explicitly included in the expression.

- This is an example of a more complex expression:

  nvl(pseudo_sale,nvl(shipments,orders))

  Here user overrides take precedent. If no user overrides are present, the expression uses shipments where available, otherwise it defaults to orders.

- To use a different demand stream such as booking, enter the following expression:

  nvl(pseudo_sale,actual_quantity)*(1 + nvl(demand_fact,0))

- For a Spares Forecasting demand stream (not based on the SALES_DATA table), the following expression represents a possible configuration:

  nvl(nvl(spf_shipment_over, spf_shipment_in),0)

6. Click Save.

> **Note:** When specifying quantity_form definitions, it is strongly recommended that the expression be constructed such that it prevents negative values.

Maintaining Engine Versions

The engine maintains the quantity of most recent engine versions defined in system parameter profile_forecasts_versions. For each of these versions, a full complement of columns is kept in SALES_DATA and PROMOTION_DATA. If a specific Engine Profile does not have a profile_forecasts_versions parameter, then the parameter value found in the system parameter active_forecast_versions is inserted into the profile during the engine run.

To define the number of engine versions maintained:

1. Navigate to system parameters.

   Business Modeler > Parameters > System Parameters.

   The System Parameters window appears.

2. Select the Engine tab

3. Select the forecast from the Engine Profiles drop-down menu.

4. Navigate to the Time sub tab.

5. The parameter named profile_forecasts_versions, located on the Time sub tab controls the number of forecast versions for the given Engine Profile. The default value is 5.

Illegal Characters in Demantra

Within Demantra, do not use the following special characters:

Single quote (')

Double quote (")

Ampersand (&)

If you use these characters, unexpected results may occur.

2

Basic Concepts

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter introduces the basic concepts involved with configuring the Analytical Engine.

This chapter covers the following topics:

- Overview of Forecasting
- Causal Factors
- Promotions (PE Mode Only)
- Forecasting Models and the Engine Flow
- The Forecast Tree
- Influence and Switching Effects (PE Mode Only)
- Combination-Specific Settings
- The Forecast Data

Overview of Forecasting

The Analytical Engine generates a forecast that considers the historical demand and the causal factors.

In this process, the Analytical Engine calculates a set of *coefficients* that describe how each causal factor affects demand for each item-location combination, over time. The Analytical Engine then uses those coefficients, along with future values for the causal factors, to determine the forecast.

You do not see or work with the coefficients directly, but you may find it helpful to see the general equation to which they apply:

D = constant + A1*CF1 + A2*CF2 + A3*CF3 + ...

Where:

- D is the demand for a specific combination.
- constant is the constant demand for that combination, independent of time.
- CF1, CF2, CF3, and so on are the causal factors in the system. Some of them are local and apply just to this combination; others are global. All of them vary with time.
- A1, A2, A3, and so on are the coefficients that the Analytical Engine calculates for this combination. These are the same for all dates.

Demantra uses an equation like this for each combination. The Analytical Engine solves all the equations simultaneously and calculates the coefficients, which it then uses to generate the forecast.

After the forecast is generated the following information may be available:

- Base forecast
- Lift Forecast
- Item node, Location node, and the Level ID for the forecast
- Models used successfully for the forecast
- Models, which the engine attempted to use for the forecast and failed
- How the forecast was generated
- Metrics demonstrating quality of the forecast

Causal Factors

Causal factors provide information about historical events that are expected to recur in the future. Causal factors cause demand to deviate from a trend. More specifically, a causal factor is a time-varying quantity (such as price, season, or day of the week) that affects demand. Demantra requires historical data for causal factors, as well as future data that describes expected occurrences that will affect demand.

> **Note:** The Analytical Engine uses multiple theoretical models, and not all of them consider causal factors; see "Forecasting Models and the Engine Flow".

Types of Causal Factors

Demantra uses the following general types of causal factors:

- *Global causal factors (global factors)* apply to all item-location combinations. For example, a season is a global causal factor. Most Demantra implementation use global factors. Oracle provides a set of base causal factors; see "Base Causal Factors".

- *Local causal factors* apply to specific item-location combinations. For example, a discount applied to a specific item in a specific sales region is a local causal factor. Price is another local causal factor.

  Local causal factors include activities, which are a special kind of local causal factor that supports *activity shape modeling*; see "Activities and Activity Shape Modeling".

- **(For PE mode only)** *Promotional causal factors* apply to specific item-location combinations *and* to specific promotions. Promotional causal factors are available only within Promotion Effectiveness. Promotional causal factors are based on the attributes of the promotions in the system. You can use promotional causal factors to perform *promotional shape modeling*. See "Configuring Promotions and Promotional Causal Factors".

Base Causal Factors

Demantra provides the following base causal factors. Except for Price, these are all global causals; Price is local:

- Constant
- t (time)
- Causal factors that correspond to the months of the year. The names of these causal factors depend on the time resolution:
  - d1, d2, ..., d12 (if the time resolution is monthly or weekly)
  - m1, m2, ... m12 (if the time resolution is daily)
- Causal factors that correspond to the days of the week (included only if the time resolution is daily): d1, d2, ..., d7
- Price

For these causal factors (except Price), Demantra provides data (for many years in the future) and the correct configuration. You should not edit or delete these causal factors. In the case of Price, you need to make sure that the sales_data table contains the price information that this causal factor uses.

Data and Configuration Details

Demantra requires the following information for each causal factor:

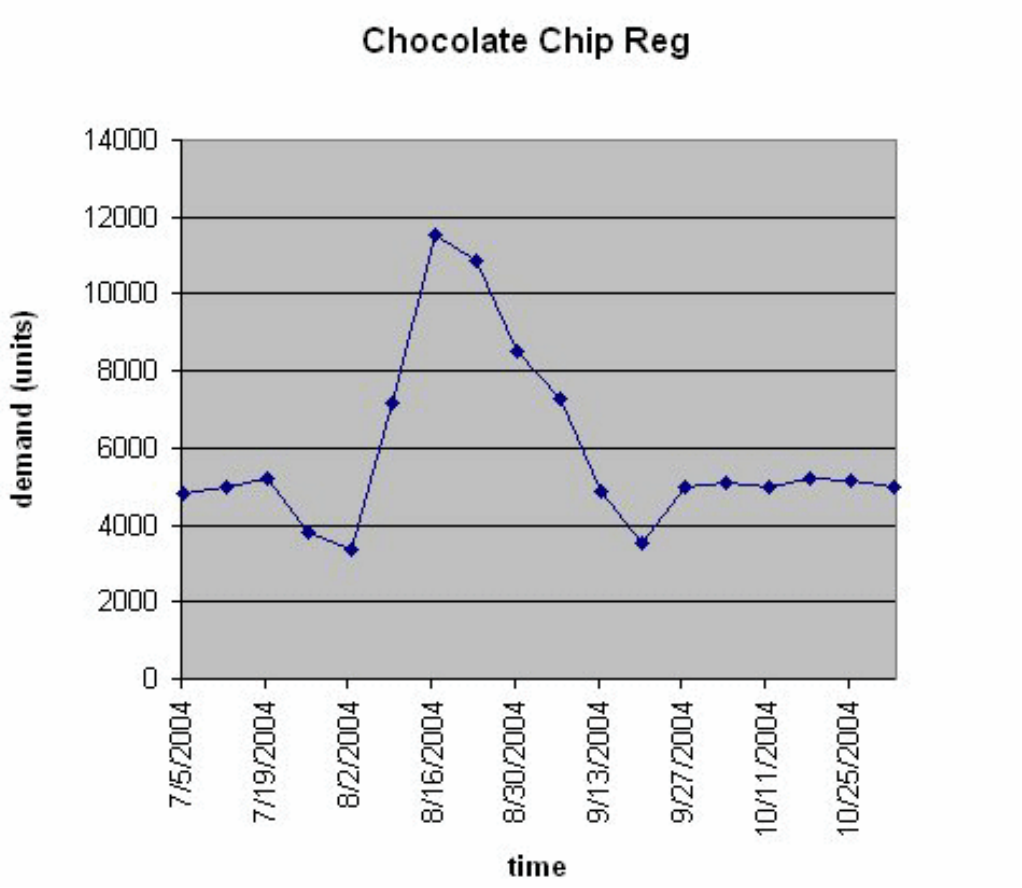
- Data for the causal factor for each time bucket, past and future.
- Configuration details on how the Analytical Engine should use this causal factor. Here you make the causal factor known to the Analytical Engine, and you specify how the engine should use it.

For reference, the following table summarizes where this information is stored:

Causal factor type	Location of data	Configuration details
Global factors	Column in Inputs table	Causal Factors screen of the Forecast Tree Editor
Local causal factors other than activities	Column in sales_data table or SQL expression that aggregates data from that table	
Activities	Column in sales_data table	
Promotional causal factors (For PE mode only)	Aggregation function retrieves data from the promotion_data and promotion tables	Promotional Causal Factors screen of the Forecast Tree Editor

Activities and Activity Shape Modeling

The Demantra *activity shape modeling* feature helps you easily reapply a demand profile that has a distinct shape over time. For any causal factor, Demantra requires past and future data. In the case of causal factors such as price and seasons, it is a simple process to obtain and load the data. Other causal factors are more difficult to describe mathematically. For example, when you run a promotional activity on a product, you may see a demand curve like the following:



If you plan a future activity that is similar to this historic activity, you would expect it to create similar demand. In general, shape modeling lets you do the following:

- Identify a historic demand curve as a reusable curve
- Create another instance of that curve starting at some future date, creating a new activity

Demantra internally represents the shape as a linear combination of as many as eight Oracle proprietary shapes. Then the Analytical Engine automatically uses this demand shape along with all the other data in the system to determine the forecast.

By default, the Analytical Engine averages the most recent data for a given shape with the stored information about that shape, which is an average of all the past observations of this shape. Users can control this, by forcing the Analytical Engine to rescale the generated shape to align with the recent data. Specifically, the user can indicate the number of buckets for which the shape alignment should occur, starting with the beginning of the shape. Typically the user specifies either 0 (the default) or the length of the shape (to realign the entire shape).

Note: Shape modeling capabilities are different in the two engine modes:

- In DP mode, the engine supports only activity shape modeling.
- In PE mode, the engine supports both activity shape modeling *and* promotional shape modeling. See "About Promotion Shape Modeling".

See "Engine Modes: DP and PE".

See also

"Configuring Causal Factors"

"Configuring Promotions and Promotional Causal Factors" (PE only)

Promotions (PE Mode Only)

A promotion is an occurrence that starts at a specific date, has a certain duration, and has a certain time-varying affect on sales. Specifically, within Promotion Effectiveness, a promotion is associated with one or more item-location combinations (at any aggregation level) for a given time bucket or buckets. A given combination can have multiple promotions at any given time bucket.

As with sales data, promotion data can be imported. Depending on how your system is configured, Promotion Effectiveness may continue to import new promotions or users might create promotions within the Promotion Effectiveness user interface. Promotion Effectiveness displays promotions in the Activity Browser in the worksheets; here users create, edit, and remove promotions.

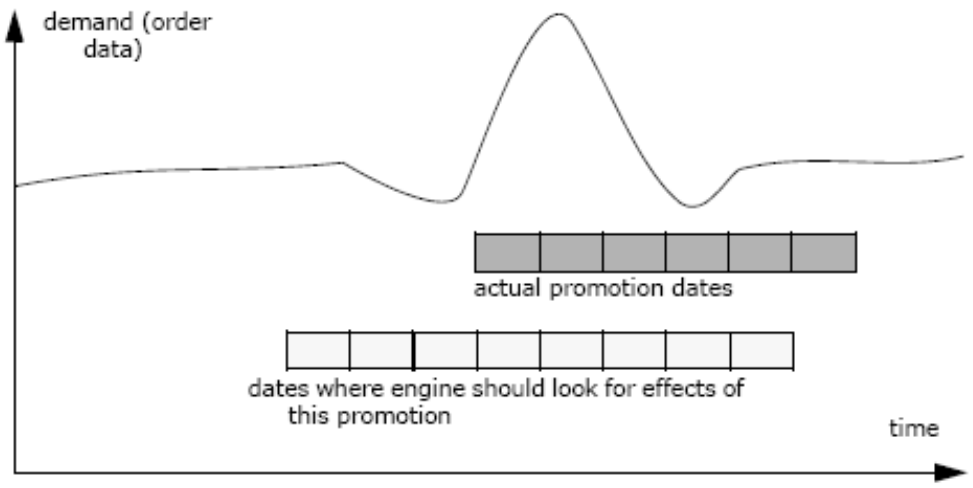
Promotion Attributes

The Analytical Engine does not use the promotions directly. Rather it uses the promotion attributes, such as discount amount, display type, and so on, each of which can have a different effect on demand. The Analytical Engine converts the values of the promotion attributes to promotional causal factors.

During implementation, you specify the attributes that promotions can have, and you specify rules for converting attribute values into causal factors. When users create promotions within Promotion Effectiveness, they specify values for these attributes.

Promotion Dates

Promotion Effectiveness assumes that a promotion has an effect between its start and end dates, as provided to Demantra, but typically the promotion has an actual effect in a slightly different span of time, as in the following example:



There is often a lag between the demand and the promotion associated with that demand. Typically this lag is larger with order data than with point of sale (POS) data, because retailers place orders further in advance. But there is often a lag even with POS data because customers know about an upcoming promotion and often delay normal purchases until the promotion occurs.

Accordingly, Promotion Effectiveness supports a couple of adjustments:

- First, you can specify an overall shift, which forces the Analytical Engine to shift the promotion dates globally by a specific number of time buckets. In the example above, the shift is -1 bucket.

  This shift time applies to all the promotions (but not to other causal factors).

- Second, you can lengthen or "stretch" the span of a promotion by specifying an additional number of time buckets on either end of the promotion. In the preceding example, we added two time buckets to the start of the promotion.

  > **Note:** Users may want to add lift or other overrides to the promotion. It is important to remember before the Analytical Engine has been run, the database contains records only for the actual promotion dates; these records are created when the promotion is created. So overrides can be entered only on those dates.
  >
  > After the engine has been run, however, the database has records for the additional dates as well and overrides can then be entered.

Promotion Hierarchy

For the benefit of users who are creating or managing promotions, you can provide a

hierarchy that helps the users group the promotions. Then, within a worksheet, the Activity Browser can display that hierarchy, as in the following example:

The Analytical Engine ignores the hierarchy itself. For the engine, the important consideration is the promotion attributes, as noted earlier.

Promotions and Promotion Shape Modeling

In addition to performing shape modeling for activities, the Promotion Effectiveness supports *shape modeling* for promotions. Specifically, you enable shape modeling for individual promotional causal factors, as needed.

As with ordinary activity shape modeling, Demantra internally represents the shape as a linear combination of the shapes. Then the Analytical Engine automatically uses this demand shape along with all the other data in the system to determine the forecast.

Forecasting Models and the Engine Flow

The Analytical Engine uses a set of theoretical models, each of which evaluates some or all of the data. Most, but not all, of these models use causal factors. The models are documented in "Theoretical Engine Models".

The Analytical Engine follows a specific process of examining the data, checking for outliers and so on, evaluating the usefulness of each theoretical model, and generating the forecast. This process is described in detail in The Forecasting Process, page 9-6.

Demantra supports multiple Analytical Engine profiles. These engine profiles can be configured to generate forecasts for different scenarios such as service parts planning. These engine profiles are described in detail in Engine Profiles, page 1-4.

Demantra provides parameters to control both the theoretical models and the overall engine flow. See Tuning the Analytical Engine, page 7-x; only advanced users should adjust these parameters.

The Forecast Tree

In general, forecasting is most accurate when it can be performed at the lowest possible allowed aggregation level. However, sometimes there is not enough data at that level for all combinations. For those combinations, the Analytical Engine aggregates the data to a higher level and tries to generate a forecast there. The purpose of the forecast tree is to organize data for this process.

Note: The Analytical Engine also considers flags on different combinations; see "Combination-Specific Settings".

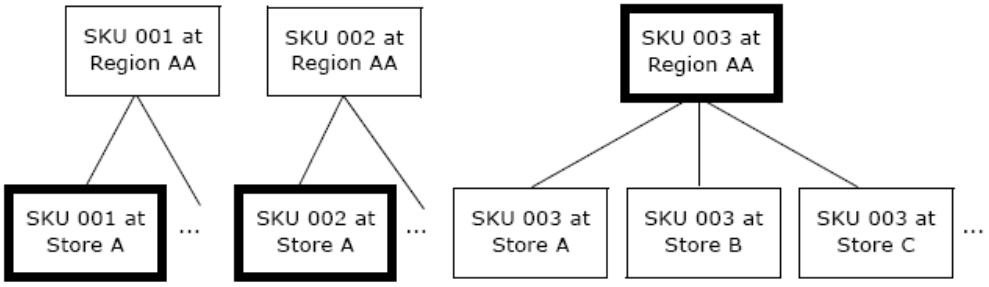
A forecast tree is associated with each of the configured batch engine profiles, thereby providing unique forecasting results for different forecast scenarios such as service parts planning.

As noted in "Levels", you define aggregation levels for use in worksheets. You use some of these levels to build the forecast tree. For PE mode, you also use the forecast tree to define the influence relationships.

Basics

Whenever the Analytical Engine generates a forecast at an aggregate level, it automatically splits the forecast for the parent node across the child nodes, again using the structure of the forecast tree. The proport mechanism controls how the aggregated forecast is split. For information on tuning proport, see "Proport Mechanism".

Each node in the forecast tree aggregates both by items and by locations. The following example shows a small part of a forecast tree.



The bold boxes show the nodes at which the Analytical Engine is forecasting.

- In this example, there is enough data at the SKU-store level for SKU 001 and SKU 002; the Analytical Engine can generate a forecast at that level for those SKUs.
- On the other hand, there is less data for SKU 003, so the Analytical Engine aggregates data for that SKU across all the stores in Region AA, generates the forecast for those SKUs at the SKU-region level, and then splits to the store level.

Accuracy Metrics for Forecasts

While generating the forecasts, the Analytical Engine also generates the accuracy metrics for the forecast to provide information regarding the quality of the forecast. The Analytical Engine generates the quality measures for forecasted combinations based on analysis of past forecasts, as the quality of a forecast is largely indeterminable without performing sample tests on the forecast data. Accuracy metrics generated by the

Analytical Engine are known as in-sample metrics.

In-sample accuracy metrics use the following formulas to judge the quality of the generated forecast for all spares considered in that run:

- **MAPE Mean (Absolute Percentage Error):** Represented by mean (abs (Series - Fit)) / mean (abs (Series))
- **RMSE Root (Mean Squared Error):** Represented by sqrt {sum (Resid. ^ 2) / (LengthSeries - Complexity)}
- **PBias (Percentage Bias):** Represented by sum (Resid) / sum (Series)
- **Relative_Error (Relative Error):** Represented by median (abs (Series. / Fit-1))

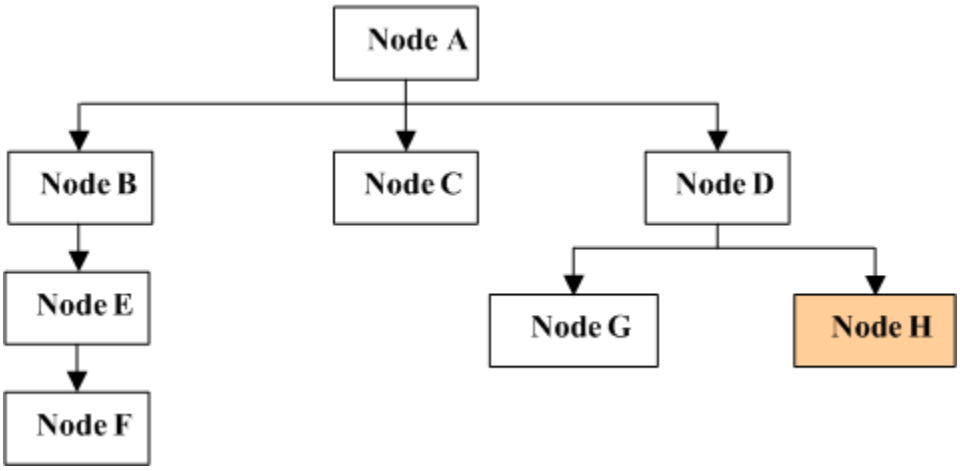
The observations made by the Analytical Engine using the above-mentioned formulas are stored in the MDP_MATRIX table.

Additional out-of-sample error calculation can be done using a stand alone error calculation process. For more details refer to *Out of Sample Error Calculations* in this document.

For a forecast tree, the accuracy metrics allocate down to the lowest level of the tree.

Example

The following figure depicts a forecast tree, where each node represents aggregated data, both by items and by locations.



Node F has a forecast at the lowest level. Therefore, all accuracy metrics generated at node F would be assigned to the member data for node F.

Node G has a forecast at the lowest level. Therefore, all accuracy metrics generated at node G would be assigned to the member data for node G.

Node H failed at the lowest level and the forecast eventually is generated at node D. The accuracy metrics from node D should be allocated to all nodes that get a forecast from node D. Node G will get the accuracy metrics from node D, whereas node H will

not receive the same from node D.

Forecast Tree Example

The following list describes a possible forecast tree.

1. Highest level: all items and all locations, aggregated together
2. All items-Division
3. Brand-Division
4. Brand-Region
5. SKU-Region
6. Lowest level: SKU-Store

Finding the Effects of Promotions (PE Mode Only)

For PE mode, the forecast tree must also be organized to support how the Analytical Engine identifies the effects of promotions. When you set up the forecast tree, you identify levels in the tree that the Analytical Engine can use in specific ways, as follows:

- The *influence range* level (IRL); see "Influence Ranges".
- The *influence group* level (IGL); see "Influence Groups".
- The lowest promotion level (LPL), which is lower than (or the same as) the IGL. This specifies the level at which promotions can be meaningfully aggregated together. For any node in the LPL, all promotions are assumed to have the same attribute values.

Out of Sample Error Calculations

Service Parts Forecasting makes use of both in-sample MAPE, see *Accuracy Metrics for Forecasts* in this document and out-of-sample MAPE, a calculated metric that occurs later, not during the batch engine run. SPF MAPE (out-of-sample) compares the actual sales data as it becomes available in the system to the forecast for a specified time period, and displays the difference as a percentage. This value is based on the forecast that is sent to Service Parts Planning (determined by SPF Final Forecast) and then archived in Demantra. It is a combination of the analytical forecast, the calculated forecast, and any user overrides.

Formula: sum (absolute value | Actual Demand - Lagged Forecast | / (Actual Demand) / Number of Observations For example, a forecast is generated monthly and the total average quantity for each of the first six months of 2010 is 2000 units, resulting in forecast of January of 2000 units Actual sales for January were lower; 1634. Therefore,

this series would display a value of 22.4% (the result of abs(2000-1634)/1634). Additional example: A forecast is generated weekly. The expected demand for the first week in March is 300, but the actual demand was 346. Therefore, this series would display a value of 13.3% (the result of abs(300-346)/346). If either demand or the forecast is zero (0), then the value would be 0%. If both forecast and demand are zero, then the value would be 100%.

The SPF Calc Forecast Accuracy seeded workflow is used to calculate error and variability associated with Service Parts Forecasting. This workflow can be called ad-hoc when accuracy measures should be generated. The seeded workflow is configured to aggregate information at levels Organization and Latest revision for the last four periods of history. The first three series pairs generate an accuracy measure for the final, analytical and calculated forecast streams by comparing the latest archived forecast with actual usage values. The last series pairs compare the last two archived versions of the final forecast with each other to determine forecast variability. If additional error calculation processes are required, it is recommended that additional steps be added to this workflow or separate workflows be created to call the APPROC_CALC_ACCURACY stored procedure.

APPROC_CALC_ACCURACY Stored Procedure

Error calculation is called from the procedure APPROC_CALC_ACCURACY. The procedure has two parameters as inputs defining the aggregation level at which the calculation is done. Each level defines an aggregation dimension. When calculating error for series on sales_data, one level from the item and one level from the location dimension should be specified. If the series being compared reside on a General Level data table, one of the levels specified can reside on the General Level. Levels should be specified by the internal ID of the level being referenced. The calculation start and end period are integers specifying which range periods should be aggregated together when calculating the error.

These parameters are relative to the current end of history date specified by the max_sales_date parameter. A value of 0 would match this date, a value of -1 would be one period prior to this date and a value of 5 would be five periods after this date. For example, if the last four periods of history should participate in the calculations, the values -3 and 0 should be assigned to the start and end parameters respectively. The series groups define the groups of three series associated with the calculation. As defined in the previous formula, the difference between series 1 and series 2 are aggregated to the defined levels and time range and then divided by the total of series one for the same aggregation. The resulting value is written to the third series which is the output series.

All three series in a group must reside on the same data model dimension, the first two residing on a data table and the third residing on the combination table associated with the two. For example, if series 1 and 2 are associated with the Service Part data table, the third series must be associated with the Service Part Matrix table. Up to five series groups may be defined. Each series group can refer to a different data table and dimension.

Inputs are:

- Calculation Aggregation Level 1
- Calculation Aggregation Level 2
- Calculation Start Period
- Calculation End Period
- Series group 1: Inputs 1, Input 2 and Output
- Series group 2: Inputs 1, Input 2 and Output
- Series group 3: Inputs 1, Input 2 and Output
- Series group 4: Inputs 1, Input 2 and Output
- Series group 5: Inputs 1, Input 2 and Output

The parameters p_use_parallel_hint and p_num_parallel_jobs define the parallel run of the APPROC_CALC_ACCURACY procedure. These parameters are used to improve performance of MAPE calculation.

The parameter p_num_parallel_jobs indicates the number of parallel jobs when MAPE is executed. The value of this parameter must be larger than 1. The maximum value of the p_num_parallel_jobs is JOB_QUEUE_PROCESSES in your Oracle database configuration.

The parameter p_use_parallel_hint can be 0 or 1. When p_use_parallel_hint is set to 0, no parallel hints will be used. When p_use_parallel_hint is 1, the parallel hints on the result matrix table will be used. The number of parallel hints in this case is equal to p_num_parallel_jobs.

Influence and Switching Effects (PE Mode Only)

To describe how the item-location combinations affect each other, you specify the influence ranges, influence groups, competitive item groups, and competitive location groups.

Influence Ranges

When you define the forecast tree, you specify the *influence ranges* (IR). To do so, you specify the *influence range level* (IRL); each node within the IRL is an influence range.

Each influence range is a set of combinations that do not interact with combinations within any other IR. The *influence ranges* control how far the Analytical Engine looks for influence when promotions are run. This determines the breadth of the causality. An influence range is a set of item-location combinations that potentially interact with each

other but not with combinations of other IRs. Typically each IR represents a different geographical area.

No information is available above the IRL to calculate effects of promotions. Therefore, if for certain nodes the Analytical Engine generates a forecast above the IRL, the forecast for those nodes includes only the baseline forecast.

Influence Groups

When you define the forecast tree, you also specify the *influence groups* (IG), which are subdivisions of the influence ranges. To do so, you specify the *influence group* level (IGL); each node within the IGL is an influence group.

Each influence group consists of an *item group* and a *location group* with the following behavior:

- An item group (I) is a set of items that relate identically to all other items. In particular, the items within an item group compete in the same way with items of other item groups. These items are interchangeable, as far as promotions are concerned. For example, suppose that an item group is diet colas. A promotion on any diet cola has the same effect on sales of non-diet colas, for example.

- Similarly, a location group (G) is a set of locations that relate identically to all other locations.

Using these definitions, the engine can calculate the following three causal factors for each lowest-level combination:

self	Influence caused by promotions on this combination.
own	Influence caused by other combinations within the same IG.
other	Influence caused by all other combinations within the IR.

The Analytical Engine uses these causal factors internally to calculate the switching effects caused by promotions.

No information is available above the IGL to calculate switching effects. Therefore, if for certain nodes the Analytical Engine generates a forecast above the IGL, the forecast for those nodes includes only the baseline forecast and the direct effects.

Competitive Item Groups (CI) and Competitive Location Groups (CL)

Typically, you also define *competitive item groups* (CI) and *competitive location groups* (CL):

- A *competitive item group* (CI) is a set of item groups that compete with each other.

For example, diet beverages could be a CI that contains the following three item groups: diet colas, diet fruit juices, other diet beverages. Non-diet beverages could be another CI.

- A *competitive location group* (CL) is a set of location groups that compete with each other.

You do not define these groups directly in the forecast tree. Instead, you set them via parameters. The Analytical Engine does not aggregate data to the CI and CL, so it is not strictly necessary to make them consistent with the rest of the forecast tree; they must of course, be at a higher aggregation level than the item and location groups, respectively.

Switching Effects

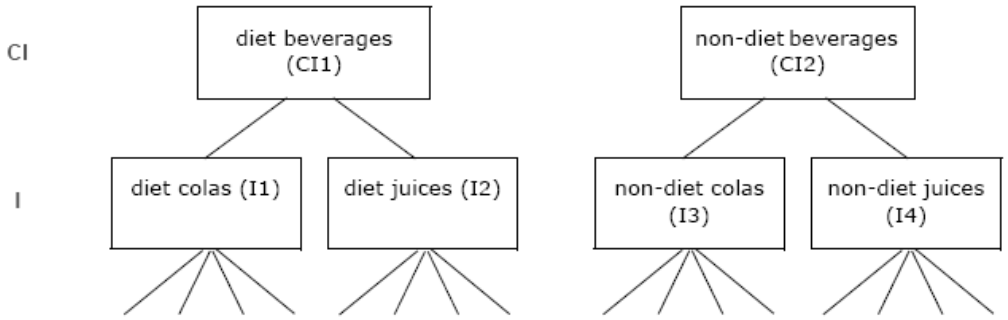
A switching effect occurs when a sale for a given item-location combination affects sales for another item-location combination. Promotion Effectiveness uses the preceding classification system to describe different switching effects. Each effect is associated with relationships between one item-location combination and others.

Effect*	CI	item group (I)	CL	location group (L)
Brand switching (or category switching)	different	*different by definition*	same	same or different
Channel switching	different	*different by definition*	different	*different by definition*
Product switching	same	same	same	same
	same	*different*	same	same
	same	*different*	different	*different by definition*
Store switching	same	same or different	same	different
	same	same	different	*different by definition*

*Depending on how you define CI, CL, I, and L, the names of these effects may or may not be appropriate. You can rename these effects by renaming the series that display them.

Notice that if the CI and CL each have only one member, there is no competition, and the only effects that can be seen are product switching and store switching.

For simple example, consider a single store and the following item groups and competitive item groups:



- If a promotion is run for a diet cola (item in I1), that can have the following effects:
- Effect on sales of other diet colas at this store. Because both items are within the same item group, this is a case of product switching.
- Effect on sales of diet juices (I2) at this store. This is another case of product switching. The items are in different item groups but are in the same CI (CI1).
- Effect on sales of non-diet colas (I3) at this store. Because non-diet colas are in a different CI than the diet colas, this is a case of category switching.

Combination-Specific Settings

The Analytical Engine also considers specific settings that vary from combination to combination, which are stored in the mdp_matrix table and which are affected by global parameters. This section provides an overview of the key settings, which are provided to support users who want closer control over the forecast. You can create levels or series that use these settings, as needed. Not all settings are meant to be changed directly.

Fictive Status

Demantra sets a flag for each combination to indicate whether that combination is real or not. In mdp_matrix, the is_fictive flag has one of the following values:

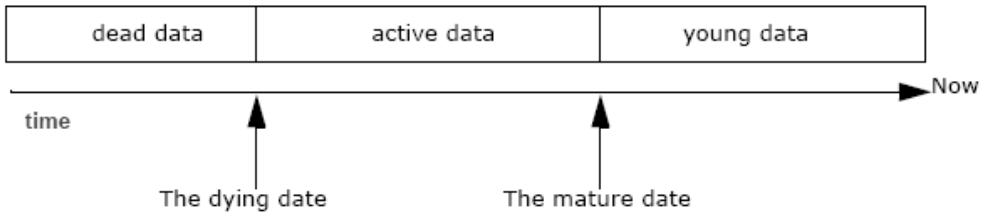
Value	General meaning
1	Combination is fictive (not real). This combination was created via Member Management.

Value	General meaning
0	Combination is real and it has non zero sales data.
2	Combination is real but all sales are zero or null.
3	Errors occurred while loading this combination.

The Analytical Engine does not use this flag directly, and users should not edit it.

Age

Each combination is either young, live, or dead, depending on the relative age of its sales data. Demantra uses two cutoff dates to determine the age of a combination:



The dying date is controlled by the dying_time parameter, and the mature date is controlled by the mature_age parameter. Both parameters are global.

Demantra automatically uses these cutoff dates as follows:

- If there are no sales for the combination after the dying date, the combination is considered dead.
- If there are no sales for the combination before the mature date, the combination is considered young.
- Otherwise, the combination is live or active.

  See "Engine Parameters".

The parameters used to determine if a combination is active, dead or young are:

- hist_glob_prop
- dying_time
- mature_age

At times there is a business need to override these global definitions. Typically this would be for specific items and locations for which a more or less reactive status change is desired. For example: The majority of combinations should be deactivated if they have not sold in three months. However, seasonal items require a different setting, and should not be deactivated unless they have not sold for more than a year. These parameters are set globally, but can be overridden individually per combination.

The User-Controlled Do_Fore Flag

Demantra provides a combination-specific flag with which advanced end users can control how the Analytical Engine works on individual combinations. This flag is in the mdp_matrix table and is called do_fore. In order to enable users to set this flag, you generally create an editable series that uses this flag. Users can set this flag to any of the following values:

Value	Meaning
0	The Analytical Engine will ignore this combination
1	The Analytical Engine will consider the combination. This is the default value.
2	The Analytical Engine will create a placeholder forecast for this combination that consists of zero values (which is useful if the user wants to create an override). The engine will otherwise ignore this combination. You typically use this setting for combinations created through Member Management.

The sole purpose of the do_fore flag is to give users a way to control the prediction status of the combination, as described next. The do_fore flag is not used directly by the Analytical Engine.

Prediction Status

In mdp_matrix, the prediction_status indicator of a combination instructs the Analytical Engine how to handle this combination. The following values are possible:

Value	Affect on the Engine	Comments
No Forecast (96)	The Analytical Engine ignores this combination.	For future use; this setting cannot currently be achieved or used.

Value	Affect on the Engine	Comments
Create Zero Forecast (97)	The Analytical Engine creates a zero forecast but otherwise ignores the combination.	
Young (98)	The Analytical Engine creates a zero forecast but otherwise ignores the combination.	Sales for this combination are too new to be used for prediction.
Dead (99)	The Analytical Engine creates a zero forecast but otherwise ignores the combination.	Sales for this combination are not recent enough to be used for prediction.
Live or Active (1)	The Analytical Engine uses this combination for forecasting.	

Demantra automatically sets the prediction_status indicator and users should not change it.

How Prediction Status Is Set for Fictive Combinations

For *fictive combinations* (is_fictive = 1), Demantra automatically sets the prediction status to 98.

How Prediction Status Is Set for Real Combinations

The proport mechanism considers all real combinations (is_fictive equal to 0 or 2). It automatically sets the prediction status indicator for each combination based on the age of the combination and the do_fore setting of the combination. Specifically, it sets the prediction_status indicator as follows:

	do_fore is 0	do_fore is 1	do_fore is 2
Combination is dead	prediction_status is 99	prediction_status is 99	prediction_status is 97
Combination is young		prediction_status is 98	

	do_fore is 0	do_fore is 1	do_fore is 2
Combination is live		prediction_status is 1	

Aggregation Flags

As noted in "The Forecast Tree", the Analytical Engine sometimes needs to aggregate lowest level data in order to create a more accurate forecast. You can control, per combination, whether this combination should be aggregated to a higher level.

- If data for a specific combination is not aggregated, then that combination is not forecasted.
- Whenever the Analytical Engine aggregates data for any combination during forecasting, the engine also splits some of the forecast to that combination, according to the stored proportions of that combination.

Demantra provides three flags so that you can specify different rules based on the age of the combination. The flags are as follows:

Flag	When used
do_aggri	If combination is live
aggri_98	If combination is young
aggri_99	If combination is dead

Each of these flags (in mdp_matrix) specifies whether to aggregate data for this combination during forecasting.

Other Combination-Specific Settings

Demantra provides many parameters that control how the Analytical Engine behaves. By default, these parameters affect all the combinations in the forecast tree. Through the user interfaces, an advanced user can set analytical parameters for individual combinations if needed.

The Forecast Data

The Analytical Engine writes the current forecast to one of the following fields in sales_data: Fore_0, Fore_1, Fore_2, and so on.

Note: For PE mode, this is the *baseline forecast*.

The Analytical Engine cycles through these columns. Each time, it writes the current forecast into one column (overwriting the oldest forecast). The Analytical Engine then adds a row to the forecast_history table that describes this forecast and that indicates which column it is stored in.

The number of saved forecasts is specified by the active_forecasts_versions parameter.

Forecast Decomposition (PE Mode Only)

For PE mode, the Analytical Engine also populates the following database fields in promotion_data to show the effects of the promotions. These fields show the effects of a given promotion on a given combination, at a given date:

Field	Purpose
fore_0_uplift	Direct lift on a combination during the promotion dates, due to a promotion specifically associated with that combination.
fore_0_pre_effect	Direct lift on this combination before the promotion dates, due to a promotion specifically associated with that combination.
fore_0_post_effect	Direct lift on this combination after the promotion dates, due to a promotion specifically associated with that combination.
fore_0_brand	Effects of brand or category switching as described in "Switching Effects".
fore_0_sw_channel	Effects of channel switching on this combination, at this date, due to this promotion.
fore_0_product	Effects of product switching on this combination, at this date, due to this promotion.
fore_0_store	Effects of store switching on this combination, at this date, due to this promotion.

For the benefit of the users, you create series that use these data fields. Be sure to

provide series names that make sense to the users and that are appropriate for the business.

Forecast Versions (for Batch Runs)

As noted earlier, Demantra keeps a number of previous forecasts (as specified by the active_forecasts_versions system parameter). The most recent batch forecast is numbered 0, the previous one is numbered 1, and so on. When the Analytical Engine generates a new forecast, it moves the previous ones to different columns in the database. See "Engine Parameters".

Each series you create is implicitly or explicitly associated with a specific forecast version or multiple forecast versions. Typically, the large majority of series are associated with the most recent forecast, but it is often useful to configure some series to capture information associated with a previous forecast, or to compare multiple forecasts.

> **Note:** If you need to display present and past versions of other data, you can configure and run *rolling data sessions*, which copy data from one series to another as specified. See "Configuring Rolling Data".
>
> For information on active_forecasts_versions, see "Non-Engine Parameters".

3

Configuring the Analytical Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to configure the Analytical Engine. It also introduces guidelines for configuring the forecast tree, causal factors, and the configure to order (CTO) feature.

This chapter covers the following topics:

- General Data Requirements
- Structure and Requirements of the Forecast Tree
- Split Forecast by Series
- Configuring SALES_DATA node-splitting
- Guidelines for the Forecast Tree
- Guidelines for Causal Factors

General Data Requirements

All the sales and causal factor data should be as complete as possible. In particular, if you do not have complete causal factor data, you may have problems like the following:

- If a causal factor does not have values for future dates, it may not have a desired effect on forecasts. For example, if the Analytical Engine has learned that changes in price have an impact on sales, and the price causal factor is not extended into the future, this implies that the future price is zero. In this case, there will be a shift in the forecast values (presumably upwards: free items "sell" well). To overcome this problem, the fill-causals method can be used by checking the fill-causal option for that causal factor.

- Likewise, if the historical data is not long enough to learn the influence of all

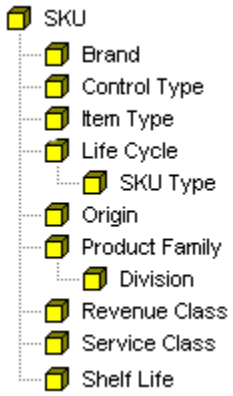
seasonal causal factors, the forecast for a missing seasonal period (for example month) may have an unexpected jump.

In general, point-of-sale (POS) data is preferable to orders. POS data is continuous, in contrast to order data, which is more sporadic, and it is easier to generate forecasts from POS data.

If you are using the Analytical Engine in PE mode, note that it is also hard to detect switching effects in order data, and the lags between promotion dates and their effects are more variable.

Structure and Requirements of the Forecast Tree

Within the forecast tree, all item levels of the tree must belong to the same item hierarchy of the same item dimension, and all location levels must belong to the location hierarchy of the same location dimension. For example, consider the following set of item levels:



Here, the **SKU** dimension includes nine hierarchies. If you include the **Life Cycle** level in the forecast tree, that means that the only other levels you can include are **SKU** and **SKU Type.** (A true implementation would have many more levels with more nesting.)

A given level can be included in multiple, adjacent levels of the forecast tree. For example, if the lowest forecast level consists of SKUs and stores, the next level above that could be the SKUs and regions. See "Forecast Tree Example".

After you specify the levels in the forecast tree, you must indicate which levels the Analytical Engine should use for various purposes, during the forecasting process. Specifically, you must indicate the following levels in the forecast tree:

Engine mode	Level	Description	Requirements
Both	Highest fictive level (HFL)	Level at which data is completely aggregated. Includes the item HFL and the location HFL.	Created automatically.
PE mode only	*Influence range level* (IRL)	Defines the influence ranges. Each node of this level is a different influence range. Typically each IR represents a different geographical area.	Must be above the influence group level (IGL). This is usually above the maximum forecast level. Oracle recommends that it is at least two levels above the IGL.
Both	Maximum forecast level	Highest aggregation level at which the Analytical Engine runs.	Must be at or above the minimum forecast level.
PE mode only	Influence group level (IGL)	Defines the influence groups. Each node of this level is a different influence group.	Must be at or above the lowest promotion level (LPL). Must be consistent with the item groups and location groups that you define in "Defining Influence and Competition (PE Mode Only)". Oracle recommends that it is two levels above the LPL.
Both	Minimum forecast level	Lowest aggregation level at which the Analytical Engine runs.	
PE mode only	*lowest promotion level* (LPL)	Lowest level at which promotions can have different attribute values from other.	Must be at or below the minimum forecast level.

Note: General levels are also valid in the forecast tree. For more

information, see "Levels".

Split Forecast by Series

Demantra time dependant information (including sales and forecast data) is always stored at the lowest possible item, location, and time granularity. And since the forecast engine is typically run at a higher level, forecast allocation or "splitting" must occur. There are two ways the engine can split higher level forecast to the lowest levels. The "matrix proportion" method splits node data according to calculated monthly proportions. These proportions are based on average historical monthly sales.

If the sales history for Item A and Item B, as a percentage at a given location, has averaged 70/30 (70% Item A, 30% Item B), then by the matrix proportion method, when forecasted together in aggregate, the engine will allocate a future forecast for these items using the same proportion. However, there are circumstances where this method is not appropriate. Some items may have no sales history, or require more granular and varying allocation rules than the monthly proportions would allow.

If new item models are replacements of existing models, a sales manager may want to modify the forecast allocation from what those previous sales histories would dictate. For example, the new Item B may have significant improvements over the old one, and the expectation is that a split between the two would instead be 40/60. The ProportionSeries parameter allows users to modify forecast allocation rules for this kind of situation. It allows you to use a "series-based proportion" method for splitting higher level forecast. The user must first choose which series will be used as provided by the forecast allocation logic. In addition, users must explicitly specify which combinations are to use the Series Based proportions. In cases where an aggregated forecast has both Matrix and Series based proportions, Matrix based proportions will occur.

Configuring SALES_DATA node-splitting

1. Open the Business Modeler and go to the Engine > Proport subtab.
2. Set the ProportionSeries parameter to the internal name of a series that will be used for apportioning node-splitting
3. Open a worksheet that includes the population for which a series-based split is desired, and add the series Engine Proportion Method to the worksheet.
4. Change the value of the series Engine Proportion Method from "Matrix Based Proportions" to "Series Based Proportions".
5. Save Updates.

Guidelines for the Forecast Tree

When creating a forecast tree, it is important to consider the following guidelines.

- The forecast tree should include an appropriate number of levels that can be forecasted.

  The forecast tree should contain 3 to 6 levels on which the engine can traverse and forecast. This number does not include any levels below the minimum forecast level and does not include the HFL.

- The forecast levels should be meaningful to the business.

  The levels of the forecast tree need to have meaningfully changing data sets per level in order to be effective. A move from level to level should substantially increase the amount of data that is being analyzed by the Analytical Engine while maintaining an aggregation method that makes sense from a business perspective. A good guideline is to have each parent node aggregate between 3 to 12 lower level nodes (on average).

- The minimum and maximum forecast levels should contain reasonable and relevant data.

  The minimum forecast level should have enough data to facilitate a forecast desirable by the customer. For instance, if exponential smoothing is not desired, then try to ensure that the lowest level has a long enough sales history for a non-exponential smoothing model to be active.

  The maximum forecast level should still be disaggregated enough to maintain some data granularity. As a general rule, none of the maximum forecast level nodes should contain more than five percent of the total data set; this means the maximum forecast level should have at least 20 nodes, and perhaps more.

- It is useful for the forecast tree to include the level on which accuracy is measured, if possible.

  Accuracy is often measured at a specific level. Often the best results can be seen if the forecast is also done at this level. This is not always true or possible but the option should be seriously considered.

- The TLMO (the level just below the top level, called top level minus one), affects performance, because it is the level for which the Analytical Engine generates the sales_data_engine table. (In the case of the Distributed Engine, Demantra creates multiple, temporary versions of this table.) As a consequence:

  - When you are using the Distributed Engine, each engine task (distributed process) receives one or more nodes of the TLMO. In order to take best advantage of the distributed mode, it is advisable for the TLMO to have many nodes and to ensure that none of them contains too many lowest level

combinations.

- If the nodes of the TLMO are comparatively small, the Analytical Engine generates sales_data_engine more quickly, which reduces run time.

- If the nodes of the TLMO are comparatively small, simulation can run more quickly, for *two* reasons: because the Analytical Engine searches smaller amounts of data and because sales_data_engine is generated more quickly.

- When you plan the forecast tree for PE mode, consider how you will set the LPL, IGL, and IRL. It is generally good to have a large number of influence ranges, each of which has a relatively small number of influence groups. Because the effect of promotions cannot be evaluated above the IRL, that means the IRL should be a fairly high level in the tree. To minimize the number of influence groups per influence range, the IGL should be fairly close to IRL.

Guidelines for Causal Factors

- It is important to avoid introducing too many causal factors, for mathematical reasons. For a given combination, if Demantra has more causal factors than sales data points, then it is mathematically impossible to calculate the coefficients for that combination. And as you approach the mathematical limits, the computation becomes progressively more difficult.

  It is desirable to have a ratio of about 3 to 5 data points per causal factor. For mathematical reasons, you must have at least 2 more data points than causal factors for any given combination.

  For example, in a monthly system, if you have two years' worth of data, that represents about 24 data points (maximum) for any combination. It would be desirable to have no more than 8 causal factors for any combination.

  It is useful to count up the causal factors you plan to use and to discard any that are not truly needed, if the count is too high. Remember that you typically need the base causal factors (see "Base Causal Factors") in addition to any other causal factors you add, so be sure to include those in your count.

- Using either shape modeling feature adds causal factors, so consider carefully when to use these features. When you model a causal factor as a shape, the data for that causal factor is replaced by as many as eight shapes internally. Each internal shape is a causal factor. You can limit the number of shapes that the Analytical Engine uses internally.

- Shape modeling generally requires continuous data: POS data rather than orders. Each shape that you model should also be multiple time buckets in length; otherwise, there is no real shape to use.

- The causal factors should not be co-linear; that is, they should not have a significant degree of dependence on each other. If the causal factors are co-linear, that introduces numerical instability and the Analytical Engine can produce unreliable results.

Additional Guidelines for PE Mode Only

- Note that when you transpose a promotional causal factor (such as a qualitative attribute), that creates additional causal factors (one for each value of the attribute). See "How the Analytical Engine Uses Promotions".

- As you create promotional causal factors, consider maintenance issues. You may not have complete control over the original location and form of the promotional data, and you may need to write procedures to maintain the promotional tables that the Analytical Engine uses.

- Pay attention to the order in which the Analytical Engine processes the promotional causal factors, which can have an impact on performance. For example, if you need to filter data, it is better to do so earlier rather than later. See "How the Analytical Engine Uses Promotions".

4

Configuring the Forecast Tree

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to configure the forecast tree. In the case of PE mode, it also describes how to configure the influence relationships, and competition among the combinations.

This chapter covers the following topics:

- Configuring the Forecast Tree
- Pooled Time Series
- Defining Influence and Competition (PE Mode Only)
- Defining the Forecast Tree for Service Parts Planning Supersessions
- Specifying Additional Parameters

Configuring the Forecast Tree

Caution: The Promotion Optimization engine uses levels in the Oracle Demantra forecast tree, and uses their names rather than their internal identifiers. This means that if you change the name of a level, you must rebuild the forecast tree to make sure that Promotion Optimization can find the level (because the forecast tree is not automatically synchronized with the level definitions).

If you are not using Promotion Optimization, you would need to rebuild the forecast tree only if you remove a level or add a new level that you want to include in the forecast tree.

See also

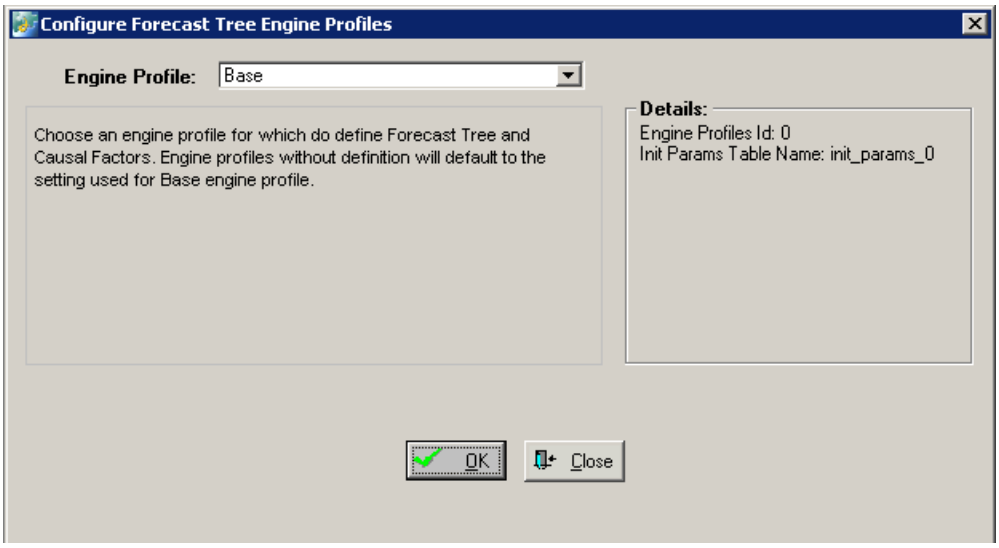
"Basic Concepts"

"Guidelines for the Forecast Tree"

To configure the forecast tree:

1. Click Engine > Forecast Tree. Or click the Forecast Tree button.

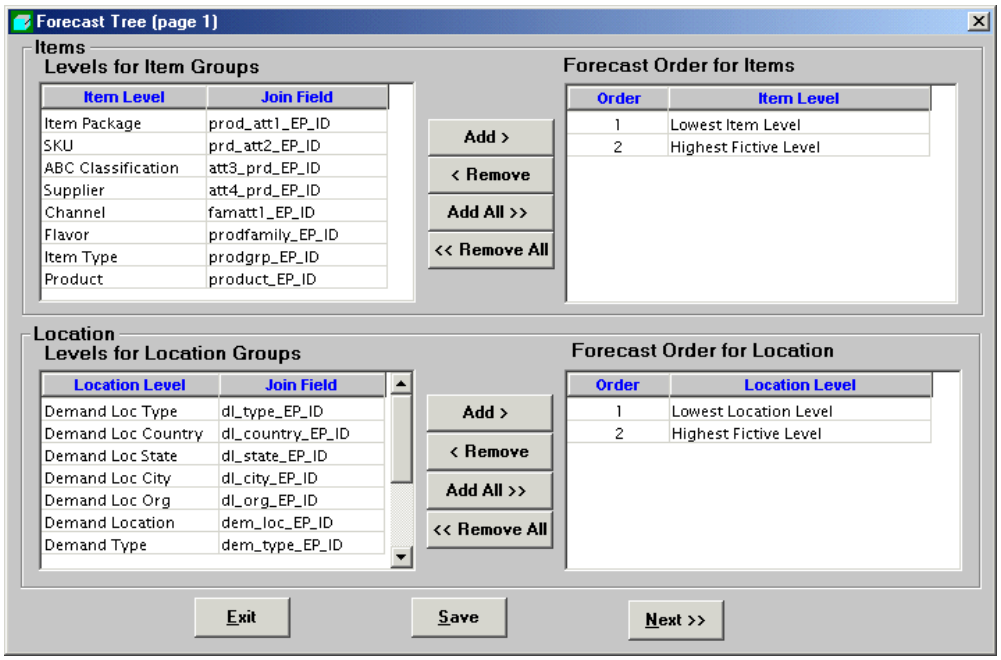
   The Configure Forecast Tree Engine Profiles dialog box appears.



2. Select the engine profile for the forecast tree you want to modify and click OK.

   The Forecast Tree Editor displays lists of all the item and location levels that you have created in the system.

   > **Note:** General levels can also be selected in the forecast tree instead of an item or location level. For example, in the case of service parts planning, the general level "Lowest spf Level" and "SPF Latest Revision" are selected as item levels. For general levels to be available in forecast tree configuration, the engine profile must refer to a general level data table. The engine parameter EngDimDef_ItemOrLoc determines whether the general levels will appear in the item or location dimension of the forecast tree.



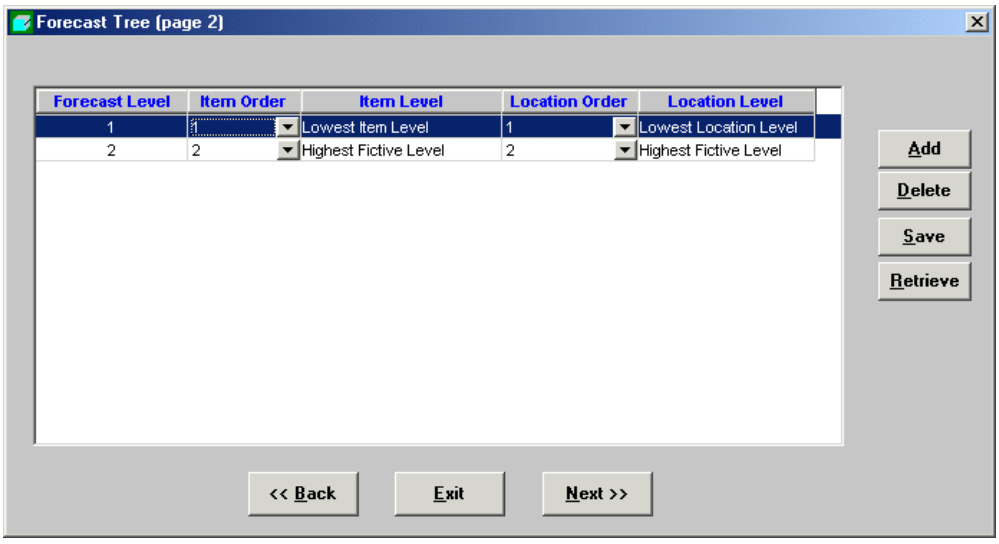
You use this dialog box to select the item levels and location levels to include in the forecast tree.

> **Note:** *As you select item and location levels in the following steps, add levels from the lowest level to the highest.* Business Modeler automatically adds the highest fictive level to each list.
>
> You can have different number of elements in these two lists.

3. Select the item levels to be included in the forecast tree. To do so, use the two lists at the top of the dialog box. Use any of the following techniques:
   - In the left list, double-click a row.
   - Click a row and then click Add.
   - Click Add All to transfer all items.
4. Select the location levels to be included in the forecast tree. Use the two lists at the bottom of the forecast tree, and use any of the methods described in the previous step.
5. When you have finished selecting levels, click Save.
6. Click Next.

   The Forecast Tree Editor displays a dialog box that you use to build the forecast tree

itself.



In this dialog box, each row corresponds to a level in the forecast tree. In turn, a level in the forecast tree consists of one item level and one location level.

> **Note:** *As you build the forecast tree, add levels from the lowest level to the highest.* Business Modeler automatically adds the HFL, if you do not do so explicitly.

7. To create a level in the forecast tree, do the following:
   1. Click Add.
   2. In the drop down list in the Item Order column, select an item level.
   3. In the drop down list in the Location Order column, select a location level.
8. Add more levels to the forecast tree as needed, and then click Save.
9. Click Exit or click Next.

   If your system includes Promotion Effectiveness, the Forecast Tree Promotion Levels screen appears. This screen displays the forecast levels as created in the previous screen.

10. (PE mode only) On this screen, specify the following:

    - Level to use as the lowest promotional level (LPL). This is the lowest aggregation level the Analytical Engine will consider when evaluating the effects of promotions.

    - Level that defines the influence groups. This is the influence group level (IGL). This indirectly specifies the item groups and location groups.

    - Level that defines the influence ranges. This is the influence range level (IRL).

    - If the system includes modules AFDM, PTP or TPO, an additional screen is available. This screen controls whether the engine simply aggregates data when forecasting at higher levels or whether it groups aggregated data nodes into longer time series.

    - As a default, the Forecast Detail and Forecast Range should be set to the same levels as those in the dialog with title "Forecast Tree (page 2)." For additional information regarding modifications of this screen see Pooled Time Series Below.

    For example, in the row that should corresponds to the influence range level, select Influence Range from the drop down list in Promotion Level Type.

    > **Note:** To establish the LPL and IGL at the same level, select the option Lowest Promotion Level & Influence Group.

11. Do one of the following:

    - Click Next. Business Modeler next displays the Causal Factors dialog box; see "Configuring Global and Local Causal Factors".

    - Click Exit. You can return later to configure causal factors.

    See also

    "Guidelines for the Forecast Tree"

Pooled Time Series

When the forecast tree calculation encounters a node for which it cannot generate a forecast, it traverses up to the next highest forecast tree node to generate the forecast. The engine then allocates an appropriate value back down, according to a proportional algorithm. For example, if a particular product/store did not have enough historical data, then the forecast engine would aggregate data at a higher level, such as product/region, generate a forecast at this level, then allocate a proportional amount of that back down to the product/store node.

One issue with this approach is that the product/region contains information that has been aggregated (summed), and aggregated data is often smoothed, resulting in less granular information being available. This can cause granular historical behavior as well as causal factor information to be factored out of the forecast. For example, the product/region node may be an aggregate of a number of different stores, and if there is specific historical pattern for that store--say, that the location always has a sales spike at a particular time of year-- then that information would become smoothed over when forecasting at an aggregate level.

Pooling a times series is a way of supplying the engine with more information than what would have been available without summing several nodes together. Instead of aggregating the data at a higher level node, it concatenates, or "pools" the data, allowing it to evaluate all the more granular data points together. The figures below show an example of these different types of data:

Raw Data

		Date										
Item	Store	1/1/2007	1/8/2007	1/15/2007	1/22/2007	1/29/2007	2/5/2007	2/12/2007	2/19/2007	2/26/2007	3/5/2007	3/12/2
Az-175	122A	99	89	120	20	107	114	84	250	80	108	
Az-175	122B	92	90	65	165	118	83	87	10	75	115	
Az-175	122C	84	117	103	140	105	74	106	40	110	120	

Aggregated Data

		Date										
Item	Store	1/1/2007	1/8/2007	1/15/2007	1/22/2007	1/29/2007	2/5/2007	2/12/2007	2/19/2007	2/26/2007	3/5/2007	3/12/2
Az-175	All	276	297	267	325	330	271	277	300	265	342	

Pooled Data

- Forecast the initial demand for a new product in an existing store (with no sales history) based on how other products pooled together at the same store
- Forecast the initial demand for existing products in a new store, based on sales at similar stores
- Better capture the lifecycle of short lifecycle products.
- Help capture the impact of a promotion type that is new to a product by leveraging its historical impact on similar products
- Capture overlapping product introductions, phase out and allocation complexities within a product family
- Better forecast new products with overlapping lifecycles

In addition to making up for limited sales histories, another key reason for using pooled

time series analysis is to increase the chance for utilizing data relating to irregular occurrences, or "events." Events can be simple actions, such as a change of a price at a store, or more rare events such as natural disasters. Each event may cause an increase or decrease in demand, and the pooled time series calculation is better able to reflect this when evaluating historical demand.

> **Additional Information:** For combinations with sufficient history it is still recommended the combination be forecasted independently and not in a pooled manner. Pooled information may average the behavior across several combinations and be less accurate than focusing on the data of a specific combination.

Configuring the Forecast Tree for Pooled Time Series

When configuring the forecast tree for Pooled Time Series, an additional configuration must be made in the forecast tree. For each forecast tree level, a Forecast Detail and a Forecast Range must be defined. Forecast Detail is the data aggregation to be used during forecast generation. For example, if a region has three stores, when setting detail to level region, the forecast node value will be the aggregation of the three stores.

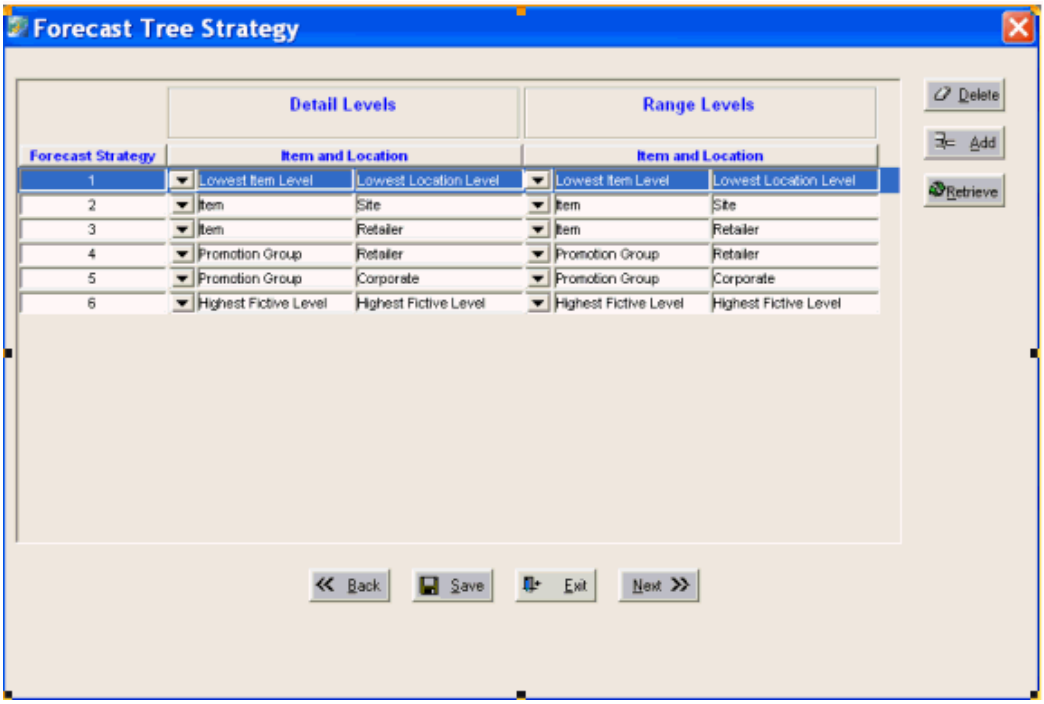
The Forecast Range defines how Forecast Detail nodes are pooled together to form larger data sets with additional information. A Forecast Range can be set to the same value as the Forecast Detail. In this case, data is aggregated to the Detail node and a forecast is generated for that node independently. If Forecast Range is set higher than Forecast Detail, more than one Detail node will be pooled and concatenated together, and then forecasted together. If the above region detail level is associated with a range level of country, then all aggregated region information will be pooled together and forecasted rather than each region-based node independently.

In the example below, we see that the Item/Site detail level is associated with two range levels. The first forecast tree level (where Range and Detail are set the same: Item/Site -- Item/Site) means that when the forecast calculation is trying to generate a forecast for the Item/Site detail level, it will generate a forecast at each Item/Site node individually. However, if the forecast fails at a specific Item/Site node, then the next forecast tree level will be used. At this next level, Range is greater than Detail (Item/Site – Item/Customer). This tells the forecast calculation to generate a forecast for Item/Site using the all the Item/Sites in the range of Item/Customer.

The following replaces Step 8 in the procedure above.

8. Click Next.

1. Click Add.
2. In the drop down list under Detail Levels, select an item and location level.
3. In the drop down list under Range Levels, select an item and location level.



Defining Influence and Competition (PE Mode Only)

To describe how the item-location combinations affect each other, you specify the following information:

- The level of the forecast tree to use as the IRL; each node within the IRL is an influence range.
- The level of the forecast tree to use as the IGL; each node within the IGL is an influence group. This indirectly specifies the item groups (I) and location groups (L).
- The level of the forecast tree to use as the LPL.
- The levels to use as the competition item groups (CI) and the competition location groups (CL). You specify these via parameters.

For the first three tasks, see "Configuring the Forecast Tree". To define the CI and CL, do the following:

1. For each level you create, Business Modeler creates a row in the group_tables table for each level. Make a note of the level ID of the levels that you want to use as the CI and CL.
2. Navigate to the Parameters > System Parameters > Engine > Shell parameters. Each

value should be a level ID as given in the group_tables table.

Parameter	Purpose
COMPETITION_ITEM	Specify the level whose members are the competitive item groups.
COMPETITION_LOCATION	Specify the level whose members are the competitive location groups.

The CI and CL should be consistent with the item groups and location groups. Specifically, any lowest level items within a given item group must belong to the same competitive item group. The easiest way to follow this rule is to set the CI equal to an item level that is higher than I and that is within the same hierarchy. A similar rule applies for the locations.

See also

"Switching Effects"

"Guidelines for the Forecast Tree"

Defining the Forecast Tree for Service Parts Planning Supersessions

Service Parts Forecasting supports the superseding of old parts with new parts, know as supersessions in EBS Service Parts Planning. The Forecast Spares Demand engine profile has been defined to use this functionality. In particular, the Forecast Spares Demand engine profile forecasts on the t_ep_spf_data table instead of SALES_DATA. In addition, service parts forecasting also refers to two engine parameters to configure the forecast. They are:

- GLPropSuperSessionMethod: Defines the method general level proportions use to allocate proportions during supersessions. When set to the default for each period, proportions are allocated completely to the member with the latest starting date. If set to All Active Revisions for each period, proportions are allocated equally among all active members.

- EngKeyDef_Supersession: Key used to aggregate members belonging to the same supersession set. When set to the same value as EngKeyDefPK, the proport calculates proportions for each lowest-level member processed by the engine individually and no special handling of supersessions is done. If set above the level defined by EngKeyDefPK, then calculation of proportions is done at this aggregated level considered the supersession and all underlying combinations receive the same proportional values.

For more information about these engine parameters, see Analytical Engine Parameters,

page 10-1.

Specifying Additional Parameters

Use the Business Modeler user interface to set the following additional engine parameters, if needed:

Parameter	Purpose
max_fore_level	The maximum level on the forecast tree at which a forecast may be produced. Upon failure at this level, the NAIVE model will be used, if enabled. For PE mode: • This level is usually below the IRL. • Sometimes the natural top forecast level does not make a good choice of IRL, and a more aggregated level would be better for the IRL. This new level may be too high for forecasting, but it is useful for calculating indirect effects. In such a case, set max_fore_level to the highest level to use for forecasting, and the IRL to the higher level.
min_fore_level	Minimum forecast level that the engine will forecast. From that level down, the engine will split the forecast using the precalculated proportions in the mdp_matrix table. The engine does not necessarily create the forecast at this level. If the results are not good at this level (for a portion of the forecast tree), the Analytical Engine moves to a higher node of the forecast tree, creates a forecast there, and splits down to the minimum forecast level. As before, the engine splits using the precalculated proportions in the mdp_matrix table. For PE mode, this level must be at or above the LPL.

For information on these parameters, see "Engine Parameters".

The Forecast Tree Editor displays a dialog box that you use to build the forecast tree itself. The Forecast Tree Editor displays lists of all the item and location levels that you have created in the system.

5

Configuring Causal Factors

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes how to create causal factors, configure them, and populate them with data. It also describes the predefined causal factors provided by Demantra.

This chapter covers the following topics:

- Notes About Causal Factors
- Creating a Global Factor
- Creating a Local Causal Factor
- Configuring Global and Local Causal Factors
- About Activity Shape Modeling
- Enabling Activity Shape Modeling
- Deleting a Causal Factor

Notes About Causal Factors

For each causal factor, you must provide data for all time buckets, both historical and in the future. Depending on the type of causal factor, this data is stored in different locations in the database. Causal factors are associated with a specific batch engine profile.

Causal factor type	Location of data	How to edit the table
Global factors	Column in Inputs table	Business Modeler Third-party database tool

Causal factor type	Location of data	How to edit the table
Local causal factors other than activities	Column in the sales_data table or SQL expression that aggregates data from that table	Third-party database tool
Activities	Column in the sales_data table	Third-party database tool

Causal Factors and Engine Models

The Analytical Engine uses a set of theoretical models, each of which evaluates some or all of the data. When you configure a causal factor, you specify the following flags to specify which models should consider that causal factor:

Flag *	Meaning
short	For use by the short models (BWINT, IREGR, LOGREGR, LOGISTIC, and REGR). These models use all causal factors that they are given.
long	For use by the long models (ARLOGISTIC, CMREGR, ELOG, ICMREGR, and MRIDGE). These models examine all the causal factors they are given, but choose the ones that give the best results.
non seasonal	For use by the non seasonal models (ARIX and ARX). The only causal factors that should be flagged as non seasonal are ones that are not a predictable function of time. For example, price varies with time, but randomly, so price should be flagged as non seasonal.
multiplicative group 1	For use only by the DMULT model. If you are using this model, each causal factor should use one of these flags.
multiplicative group 2	Typically you place causal factors that vary on a daily basis into one group and place all others in the other group. No causal factor should be in both groups. See "Theoretical Engine Models".

* Name of flag as displayed in the Causal Factors screen.

Not all models use these flags. Models not listed here do not use causal factors.

Typical Flags for Causal Factors

Typically you initially flag causal factors as follows:

	Causal Factor	Short	Long	Non-Seasonal	Multiplicative Group 1	Multiplicative Group 2
base (predefined) causal factors	CONSTANT	yes	yes	yes	no	no
	t	yes	yes	no	yes	no
	d1, ... d12* or m1, ... m12**	yes	yes	no	no	yes
	d1, ... d7**	yes	yes	no	yes	no
	price	yes	yes	yes	yes	no
your added causal factors	If factor is a predictable function of time	usually not	yes	no	if factor varies by day	if factor varies by month
	If factor is not a predictable function of time	usually not	yes	yes	no	no

*Included only if time resolution is monthly or weekly.

**Included only if time resolution is daily.

Important: In many cases, these flags have to be adjusted. Contact Oracle for assistance.

Creating a Global Factor

A global causal factor has time-varying data that applies in the same way to all items and locations.

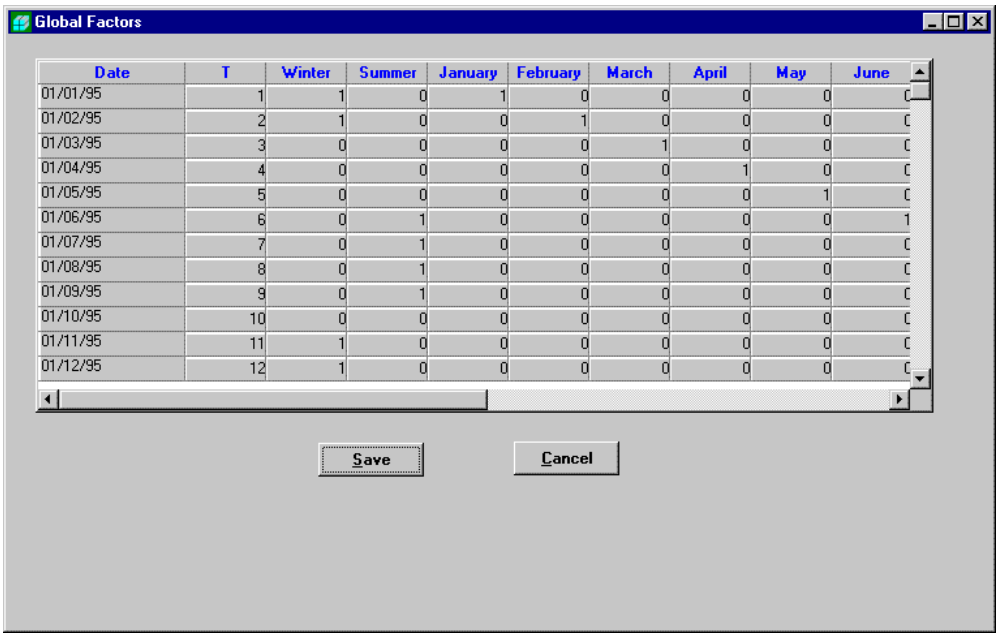
To create a global causal factor:

1. Do one of the following:
   - Go into the database and add a column to the Inputs table.
   - Create the global causal factor within the Business Modeler user interface, as follows:
     1. Click Data Model > Global Factors > Options to access the global factor user interface.
     2. Click Data Model > Global Factors > New Factor. The New Factor dialog box appears.
     3. Type in the factor name.
     4. Click Add New Factor.
     5. Click Cancel to close the dialog box. The Business Modeler adds a new column to the Inputs table.
2. Load data into the new column by using a script, a database tool, or the Business Modeler.

To use the Business Modeler, do the following

3. Click Data Model > Global Factors > Options.
4. Click Data Model > Choose Factor. Or click the Choose Factor button.
5. The Choose Factor dialog box appears.
6. Check the check box for each of the causal factors you wish to view. Make sure that Date is selected so that you can see the dates along with the causal factor data.
7. Click OK.
8. Click Data Model > Global Factors > View. Or click the Create View button.

   Business Modeler displays a table that shows the value of each global factor over time. This table displays one row for each base time bucket in the planning horizon. Each column corresponds to one global factor.



9. Select the cell or cells to be edited. The editable cells are colored white. When selected, the cells turn yellow.

10. Click Data Model > Global Factors > Edit Data. Or click the Edit Data button.

    The Edit Data dialog box appears.

11. Type the number required and click OK.

    The data appears in each highlighted cell.

12. Click Save to save your changes.

13. Click Cancel to close the dialog box.

14. Configure the global factor as described in "Configuring Global and Local Causal Factors".

    See also

    Creating a Local Causal Factor, page 5-5

    "Base Causal Factors"

Creating a Local Causal Factor

A local causal factor has time-varying data that is potentially different for each item-location combination.

To create data for a local causal factor:

1. If the sales_data table does not include a column that contains the data you want to use as a causal factor, go into the database and add the desired column.
2. Load data into the new column by using a script or by a database tool.
3. Configure the new causal factor as described in "Configuring Global and Local Causal Factors".

   See also

   "Creating a Global Factor"

Setting up the price causal factor:

The predefined price causal factor uses the field item_price in the sales_data table. You should make sure that this data is available.

Transpose Function

This allows one causal to be converted into several causals using a transpose function, with different transpose values in different periods resulting in engine accepting multiple causals instead of one. Transpose function should only refer to sales_data and mdp_matrix tables. Transpose is done after information is aggregated and it is important to ensure underlying values are consistent.

Configuring Global and Local Causal Factors

Here you provide information about how the Analytical Engine should use each global and local causal factor.

To configure a causal factor:

1. Click Engine > Forecast Tree. Or click the Forecast Tree button.
2. Select the batch engine profile for which you want to configure causal factors and click OK.

   Note: The simulation engine profiles inherit the causal factor settings from the parent batch engine profile.

3. Click Next repeatedly until you reach the Causal Factors dialog box.
4. If the causal factor is not yet listed here, do the following:
   1. Click Add.

A new line is displayed.

2. Describe the new causal factor by specifying the following:

Factor Name	Depends on the type of causal factor: • For a global factor: name of an existing column in the Inputs table. • For a local causal factor, this can be the name of an existing column in the sales_data table. The factor name can also just be a name; in this case, you must specify an expression in the Local Function field. • For an activity: name of a column in the sales_data table. Business Modeler adds this column automatically if it does not yet exist.
Factor Type	Choose one of the following: • global • local • activity (a special kind of local causal factor that supports shape modeling) Do not use the event choice, which is an older implementation of the more general local choice. The price option is useful only for the predefined price causal factor.

3. Specify how the Analytical Engine should use the causal factor. To do so, specify the following values:

Short	Usually you enable this check box only for the following global causal factors: Constant, t, d1, ... d12. See "Typical Flags for Causal Factors".
Long	Usually you enable this check box for all causal factors. See "Typical Flags for Causal Factors".

Multiplicative Group 1	Enable this check box to include this causal factor in the first multiplicative group for use by the DMULT model; you should enable this check box for at least one causal factor. Typically you place causal factors that vary on a daily basis into one group and place all others in the other group. No causal factor should be in both groups. see "Theoretical Engine Models". This setting affects only the DMULT model.
Multiplicative Group 2	Enable this check box to include this causal factor in the second multiplicative group for use by the DMULT model; you should enable this check box for at least one causal factor. This setting affects only the DMULT model.
Non Seasonal	Enable this check box if the data associated with this causal factor is not known to be a predictable function of time. For example, price varies with time, but randomly, so price should be flagged as non seasonal. See "Typical Flags for Causal Factors".
Fill Causals	Specifies whether Demantra should interpolate when values are missing for a date. The missing local causal factor will receive the average of its nearest two non-missing neighbors. For example: • If the causal values are 1, missing, and 2, then Demantra replaces the missing value with 1.5. • If the causal values are 1, missing, missing, missing, and 2, then Demantra replaces each missing value with 1.5.
Scale Causals	Specifies whether Demantra should scale causal factors to match weekly values. This parameter is only used when viewing weekly data by calendar month. For more information, see Viewing Weekly Data by Calendar Month.

Trend Causal	This indicates that a causal factor is to be used to model trending. It allows trending information to be adjusted, ensuring a more accurate trend analysis, in cases where rows may be missing or omitted. This option should be selected for all casual factors used to model any sort of trending behavior.
Shape Indicator	**Only for activities.** Specifies whether Demantra should perform shape analysis and calculations on this activity.
Omit Seasonal	**Only for activities.** This option specifies whether to nullify values of the global seasonal causal factors for the time buckets during which the causal factor occurs. Specifically this refers to the causal factors d1—d12 or d1—d7 and m1—m12. For example, if you have monthly data and you omit seasonal effects for a given causal factor Promo1, that means that Demantra switches off the causal factors for the duration of Promo1. By omitting seasonal effects, you enable Demantra to capture the shape more clearly for the analysis. This option is suitable only if you expect the effect of this causal factor to be much stronger than the seasonal effects. If causal factors overlap each other, then Demantra gives precedence to the causal factor that you have flagged to omit seasonal effects.

Local Function	**Only for local causal factors.** An SQL expression that describes how to aggregate causal factor data from the lowest level data. Use one of the following SQL aggregating functions: • Min • Max • Sum • Avg Within the expression, refer to the name of the causal factor (the column name in which the causal factor is stored). Within the expression, you can also refer to fields in the mdp_matrix table. You can also include tokens of the form #FORE@<Version>#. See "Server Expression Functions and Operators".

4. Add comments to the Comments field, if desired.

5. Click Validate to check the validity of the configuration.

6. Click Save to save changes.

   Note: To return the screen to the latest pre-saved state and reset any other changes, click Retrieve.

7. Do one of the following:

   - Click Next.

     If you created any activities, Business Modeler displays a message indicating the name of the series that it automatically creates for each activity.

     For PE mode, then the Business Modeler displays the Promotional Causal Factors dialog box. See "Configuring Promotional Causal Factors".

   - Click Exit.

See also

"Configuring Promotional Causal Factors"

"Base Causal Factors"

About Activity Shape Modeling

In shape modeling, you capture the profile of the demand over the duration of a promotion. The Analytical Engine models the overall demand as a linear combination of Oracle proprietary shapes, as many as eight shapes; this information replaces the normal causal factor that would have been used instead. The Analytical Engine calculates the coefficients for each shape, for each relevant combination.

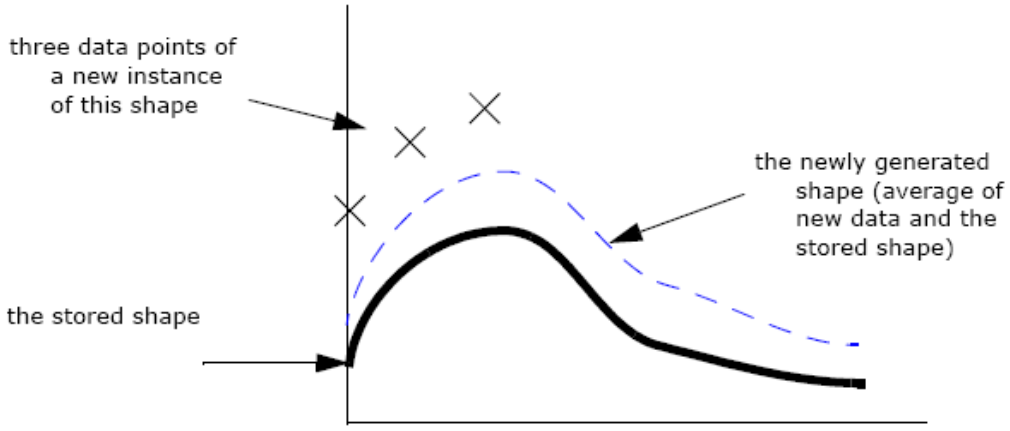
Remember that when you enable shape modeling for a causal factor, the single causal factor is replaced by up to eight causal factors. To keep the number of causal factors down, you can specify the maximum number of shapes permitted for activity shape modeling.

> **Note:** Shape modeling capabilities are different in the two engine modes:
>
> - In DP mode, the engine supports only activity shape modeling.
> - In PE mode, the engine supports both activity shape modeling *and* promotional shape modeling. See "About Promotion Shape Modeling".
>
> See "Engine Modes: DP and PE".

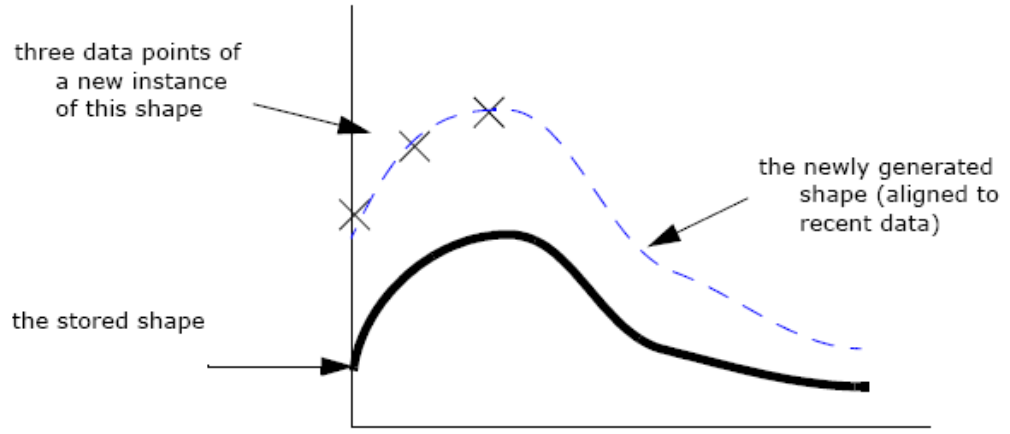
Shape Alignment

Each stored shape is an average of the past instances of that particular shape. It is important to understand that the stored *information* consists of both the shape and the actual amplitude of the curve.

When the Analytical Engine observes the beginning of a new instance of a given shape, it is necessary to decide how to set the amplitude of the new curve that it generates. By default, the engine assumes that the amplitude of the stored shape should be taken into consideration. Therefore, when the Analytical Engine generates the new shape, it averages the new data together with the stored shape, as follows:



The default behavior is appropriate when the history contains many instances of a given shape. When the shape is new to the system, however, it is more appropriate to force the Analytical Engine to re-scale the generated shape so that it aligns with the most recent observations:



To force this realignment you use the QAD (quantity alignment duration) series associated with the shape. This series specifies the number of time buckets during which this alignment should occur, starting with the beginning of the shape. If you need to align the shape, you generally should align the entire shape; that is, you set the series equal to the expected length of the shape.

Samples of Activity Shape Modeling

To see samples, use the UPGRADE_TO_SHAPE_MODELLING procedure, which does the following:

- It creates two sample activity causal factors: Product_launch and Price_change.
- It creates four series for the benefit of end users:

- Price_change
- Price_change_QAD
- Product_launch
- Product_launch_QAD

See also

"Enabling Activity Shape Modeling"

Enabling Activity Shape Modeling

To enable activity shape modeling:

1. For each specific shape you want to represent, create the causal factor data, as described in "Creating a Local Causal Factor".

2. Configure this causal factor as type Activity, as described in "Configuring Global and Local Causal Factors".

   When you configure this causal factor as an activity (named, for example, Product Launch), the Business Modeler automatically creates two series that constitute the user interface for the activity. These series are as follows:

Generic name / Example name	Purpose
Causal-factor-name/ **ProductLaunch**	Lets the user indicate the start and duration of the activity associated with a specific combination. Within this series, for each date, the user chooses "Start" or "Active" from a drop-down menu to specify the promotion start and continuation dates. The default is "None," meaning no promotion. The user identifies past activities and marks where future activities will occur.

Generic name / Example name	Purpose
Causal-factor-name_QAD/ **ProductLaunch_QAD**	Controls whether the Analytical Engine re-scales the generated shape to align with the amplitude of the most recent observed instance of this shape, for a given combination. By default, this is zero, and the Analytical Engine *averages* the most recent data with the stored shape, which is an average of all the past observations of this shape. When the shape is "new" to the system, the user should set ProductLaunch_QAD equal to the typical length of the activity, so that the new data takes precedence.

3. Add these series to a worksheet at the appropriate aggregation level.
4. Edit the *Causal-factor-name* series to identify when the activity occurred and when it will occur and save the changes.
5. If appropriate, use the *Causal-factor-name_QAD* series to control whether to realign the shape. Edit the series and save the changes.
6. To specify the maximum number of Oracle proprietary shapes that the Analytical Engine can use for activity shape modeling, set the NumShapes parameter.
7. Run the Analytical Engine as usual.

   See also

   "About Activity Shape Modeling"

Deleting a Causal Factor

To delete a causal factor:

1. Click Engine > Forecast Tree. Or click the Forecast Tree button.
2. Select the Engine Profile that contains the causal factor you want to delete and click OK.
3. Click Next repeatedly until you reach the Causal Factors dialog box or (PE mode only) the Promotional Causal Factors dialog box.
4. Click the causal factor you want to delete.

5. Click Delete.
6. Business Modeler asks for confirmation. Click Yes or No.

See also

"Configuring Global and Local Causal Factors"

"Configuring Promotional Causal Factors"

6

Configuring Promotions and Promotional Causal Factors

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

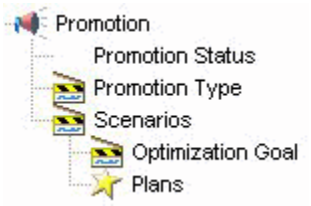
This chapter describes how to configure promotions and promotional causal factors in the Business Modeler.

This chapter covers the following topics:

- Base Behavior
- Customizing the Promotion Levels
- Loading Historical Promotions
- How the Analytical Engine Uses Promotions
- Configuring Promotional Causal Factors
- Adjusting the Promotion Dates
- About Promotion Shape Modeling
- Enabling Promotion Shape Modeling

Base Behavior

The Demantra installer automatically defines the following required promotion levels:



These levels have the following purposes:

Level	Purpose	Permitted Customization
Promotion	Defines the promotions themselves. This level must define all the possible attributes that can be associated with promotions. This level must include the Population attribute, which specifies the item-location combinations and the time span with which the promotion is associated.	Add attributes only.
Promotion Status	Controls the following: Whether the promotion is used in forecasting. Whether users can edit the promotion. How DSM uses the promotion. See "Promotion Status".	
Promotion Type		
Scenarios	Provides optional organizational structure, for the benefit of users, particularly within a worksheet.	Any change is allowed.
Optimization Goal	For use by the Promotion Optimization module only; see the Oracle Demantra Release Notes.	
Plans	Provides optional organizational structure, for the benefit of users, particularly within a worksheet.	Any change is allowed.

Promotion Status

The values of Promotion Status are as follows:

	Meaning		
Status	**In forecasts**	**In worksheets**	**In DSM**
Unplanned	The Analytical Engine does not consider this promotion.	Users can edit the promotion.	DSM considers this promotion to be uncommitted.
Planned	The Analytical Engine does consider this promotion.		
Committed		The user who committed this promotion can edit it.	DSM considers this promotion to be committed
Running		Nobody can edit the promotion.	The promotion is currently running.
Unmatched			The promotion has ended but has not yet been matched to an invoice.
Matched			The promotion has ended and has been matched to an invoice.

For information on DSM, see the *Oracle Demantra Deduction and Settlement Management User's Guide*.

Customizing the Promotion Levels

You typically customize the promotion levels by adding attributes, although other changes are also permitted; see "Base Behavior". To customize the promotion levels, you use the Business Modeler.

You should also configure the Activity Browser of the worksheets, which displays a hierarchy of promotions. The Activity Browser has the same structure for all worksheets. To configure it, you use the Business Modeler.

For information, see "Configuring the Activity Browser"

Loading Historical Promotions

To load historical promotions, use the Integration Interface Wizard, described in "Series and Level Integration". Create and execute an integration interface that loads both the

promotion members (via a level profile) and any promotion data (via a data profile).

How the Analytical Engine Uses Promotions

The Analytical Engine does not directly use the promotions for forecasting. Instead, it converts their attributes to promotional causal factors, which it then converts to normal causal factors.

In this process, it uses the configuration information that you provide in the Business Modeler. For each promotional causal factor, the key options are as follows:

- Column Name Expression
- Filter
- Transpose by Column
- Merge Function
- Aggregation Function

It is important to understand how the Analytical Engine uses these options. The following sections describe how the Analytical Engine starts with promotions and converts them to causal factors. For more details on the engine flow, see "Engine Details".

These options use expressions that refer to promotion data. Note that these expressions can refer only to the tables that are used by the levels within the hierarchy of the analytical general level (promotion). For example: promotion_data or promotion.

For information on setting these options, see "Configuring Promotional Causal Factors".

Kinds of Attributes

In this discussion, it is useful to consider the general kinds of promotional attributes that the Analytical Engine can use:

- *Quantitative attributes* such as discount. These attributes have numeric values that the Analytical Engine can use in their present form. The Analytical Engine assumes that the effects of these attributes is correlated with the value of the attribute. For example, if discount1 is larger than discount2, then discount1 has a larger effect on demand than discount2.

  The Analytical Engine does not assume that the correlation has a positive sense.

- *Boolean attributes*, where the attribute either has a value or does not have a value.

- *Qualitative attributes*, where the attribute can have one value from a given set of values. The set of values is unordered, which means that even if the values are

numeric, there is no intrinsic meaning in the relative sizes. For example, you might use numeric color codes 4 and 5, but color 5 does not have a larger effect on demand than color 4.

Demantra converts this kind of attribute into a set of unrelated causal factors. For example, color code 4 is one causal factor and color code 5 is another. For any given promotion, this causal factor either has a value or does not have a value.

Step 1: Aggregate Promotion Attributes to the LPL

The lowest promotion level (LPL) is a level in the forecast tree. Specifically, it is the lowest level at which promotions can have different attribute values from other, and it must be at or below the minimum forecast level.

The Analytical Engine retrieves the promotional causal factor data and aggregates it to the LPL, as specified by the Column Name Expression. You use an aggregating expression like the following example:

```
max(promotion_data.discount)
```

> **Note:** As with all the options discussed here, the expression can refer only to tables that are used by the levels within the hierarchy of the analytical general level (promotion). For example: promotion_data or promotion.

As a general rule, an expression that uses the max function is probably appropriate in most cases, because promotions should have the same attribute values below the LPL, by definition.

Step 2: Applying Filters

Sometimes you need to convert one set of promotional causal factor data into multiple causal factors. To do so, you use a Filter expression, an aggregating expression that evaluates to true or false. The promotional causal factor uses only the data for which the expression is true. You typically create multiple promotional causal factors, each with a different filter expression that uses a different part of the source data.

For example, consider the following promotional data. This table contains one row for each promotion for a given combination and time bucket. (For simplicity, the table shows only one combination, one time bucket, and three promotions.) The promotion_data table shows values of attributes (promo_type and discount) associated with those promotions.

Item	Location	Date	Promotion	Promo_type	Discount
100	333	1	214	1	15

Item	Location	Date	Promotion	Promo_type	Discount
100	333	1	296	2	5
100	333	1	340	3	10

Suppose that we have configured the following promotional causal factors:

Factor Name	Column Name Expression	
Special Discount	max(promotion_data.discount)	max(promotion_data.promo_type=3)
Discount	max(promotion_data.discount)	max(promotion_data.promo_type<>3)

Internally, Demantra would convert the promotion attributes to the following promotional causal factors:

Item	Location	Date	Promotion	Special Discount	Discount
100	333	1	214	0	15
100	333	1	296	0	5
100	333	1	340	10	0

Step 3: Transposing Promotion Attributes

Next, the Analytical Engine considers the Transpose by Column setting, which you use for qualitative promotion attributes. This setting converts a single promotion attribute into multiple causal factors. For quantitative or Boolean attributes, specify 0, which means that Promotion Effectiveness can use the attributes as casual factors in their present form.

For example, suppose that promotions use different "delivery types," which correspond to different mechanisms such as circulars, extra product samples, coupons, and so on. Each of these mechanisms might have a different affect on sales. Suppose we have the following example data in the promotion_data table:

Item	Location	Date	Promotion	Delivery_type
150	344	1	214	4
150	344	1	296	5
150	344	1	340	6

Because delivery_type is a qualitative attribute, it is generally appropriate to transpose it. We could configure a Delivery Type promotional causal factor, as follows:

Factor Name	Column Name Expression	Filter	Transpose by Column
Delivery Type	max(promotion_data.Delivery_type)	null	max(promotion_data.Delivery_type)

Note that Transpose by Column must be an aggregating expression.

Using this configuration data, Demantra would internally convert the preceding promotion attributes into the following set of promotional causal factors:

Item	Location	Date	Promotion	Delivery Type(4)	Delivery Type(5)	Delivery Type(6)
150	344	1	214	4	0	0
150	344	1	296	0	5	0
150	344	1	340	0	0	6

You may want to transpose by a promotion attribute (as in this example) or by members of a level in the promotion hierarchy.

> **Note:** If you use the members of a level to transpose an attribute, be sure to first filter out the default member (which has an ID of 0) of that level.

Step 4: Merging Across Promotions

Next the Analytical Engine uses the Merge Function setting, which describes how to

merge promotional causal factors that occur on the same date at the same item-location combination (thus merging across all the promotions for that combination and date).

For Merge Function, you can choose one of the functions provided by Business Modeler.

The way that you merge depends upon the meaning of the data in the promotional causal factor. For example, if you have multiple discounts on the same date, you would want to merge them by the compound rule (so that 10% and 20% are merged to 28%).

Step 5: Aggregating Attributes within the IGL

The influence group level (IGL) is another level in the forecast tree. The Analytical Engine uses this level to simplify the computational problem. It creates the following three historical promotional causal factors for each node in the forecast tree:

self	Influence on this node caused by attributes on this node
own	Influence on this node caused by other nodes within the same IG
other	Influence on this node caused by all other IGs within the IR

In this last step, the Analytical Engine uses the Aggregation Function option, which describes how to aggregate the promotional causal factor to the IGL.

> **Note:** The Analytical Engine uses the same option whenever it needs to aggregate to higher levels for forecasting purposes.

For Aggregation Function, you can choose one of the functions provided by Business Modeler.

Configuring Promotional Causal Factors

This section describes how to configure promotional causal factors. You can do most of the work within the Business Modeler, but it is necessary to go into the database for the final steps.

> **Note:** Causals participating in Optimization require these additional settings
>
> - Opti_Causal_Type
>   - 0=Value

    - 1=Price Discount %
    - 2=Spent
    - 3=Boolean
    - 4=Price Decrease $
- Opti_Causal_Type
    - 0=Value
    - 1=Price Discount %
    - 2=Spent
    - 3=Boolean
    - 4=Price Decrease $
- Opti_Causal_Output= The promotion table column containing optimized causal result.
- Opti_Transpose_Output=The promotion table column containing optimized causal transpose result.

Note: The engine only evaluates causals with direct effect. Causals with indirect effects are not evaluated.

Analytical Recommendations

- Limited Number of Causals
- Quantitative or Boolean Causals only
- Qualitative Causals supported through Transpose

To configure promotional causal factors:

1. Click Engine > Forecast Tree. Or click the Forecast Tree button.
2. Click Next repeatedly until you reach the Promotional Causal Factors screen.

   Each row in this screen specifies a promotional causal factor.
3. To add a new promotional causal factor for Promotion Effectiveness, click Add.

A new line is added.

4. Describe the promotional causal factor by specifying the following:

Factor Name	Name of the promotional causal factor. This name should consist only of alphanumeric characters.
Column Name Expression	An expression that retrieves the causal factor (promotional attribute) data and aggregates it to the LPL. For example: max(promotion_type.is_ad) An expression that uses the max function is probably appropriate in all cases, because promotions should have the same attribute values below the LPL, by definition. See "Step 1: Aggregate Promotion Attributes to the LPL".
Filter	An aggregating expression that returns the true or false value. This expression filters out promotion data that should not be used for this promotional causal factor; that is, the promotional causal factor uses only the rows for which this expression returns true. See "Step 2: Applying Filters".
Transpose by Column	An aggregating expression that returns the values by which the data is to be transposed. You usually transpose an attribute only if it is qualitative. To avoid transposing, use the value 0. See "Step 3: Transposing Promotion Attributes". **Note:** If you use the members of a level to transpose an attribute, be sure to first filter out the default member (which has an ID of 0).

Merge Function	Specifies how Oracle Demantra should internally merge promotions of the same kind that apply to the same item, location, and time. Click one of the following: • Compound (Use only for numeric causal factors. All values must be greater than or equal to 0 and less than 1; otherwise, this function throws an error.) • WAVR (Weighted average. Use only for numeric causal factors. If you use this option, also specify Merge Function Column.) • Boolean (Use for boolean causal factors or for transposed causal factors.) See "Step 4: Merging Across Promotions".
Merge Function Column	Applies only if you select WAVR for the merge function. Specifies the weights to use when performing a weighted average.

The preceding expressions can refer only to the tables that can be logically linked to levels within the hierarchy of the analytical general level (promotion). For example: promotion_data or promotion

Causal From Expression	Specifies which tables the causal will be referencing. Should include all tables referenced in Column Name Expression, Filter and Transpose.
Causal Where Expression	Should only be used if Causal From Expression is populated. Specifies logical links between tables defined in Causal From Expression to promotion and promotion_data tables. For example promotion_data.promotion_type_id = promotion_type.promotion_type_id and promotion_data.promotion_id=promotion.promotion_id

Aggregation Function	Specifies how Demantra should internally aggregate this promotional causal factor across combinations, whenever it is necessary do so. Click one of the following: • WAVR (Weighted Average. For the weights, Demantra uses the stored proportions of the combinations) • Boolean (Typically you use this if you have transposed a causal factor.) • Sum See "Step 5: Aggregating Attributes within the IGL".
Priority	Ignore this field.
Influence	Specifies the effect of this promotional causal factor on other members. Click one of the following options: • Has effect on other members (this causal factor can affect other combinations, in addition to the combinations with which it is associated) • Has direct effect only (this causal factor affects only the specific combinations with which it is associated) • Has only indirect effect (this causal factor affects only the combinations with which it is not associated)

5. Describe how the Analytical Engine should use this promotional causal factor. To do so, specify the following values:

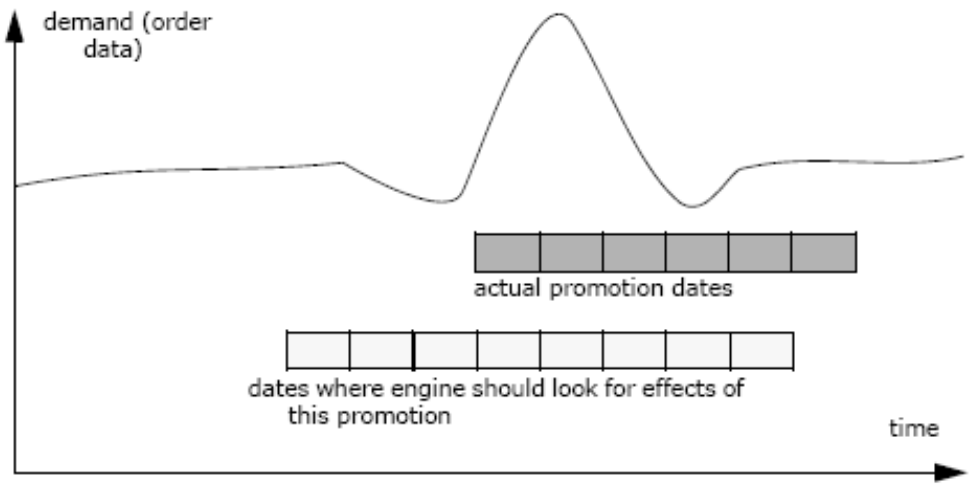
Short	Usually you enable this check box only for the following global causal factors: Constant, t, d1, ... d12. See "Causal Factors and Engine Models".
Long	Usually you enable this check box for all causal factors. See "Causal Factors and Engine Models".
Multiplicative Group 1, Multiplicative Group 2	Ignore these options, which do not affect promotional causal factors.

Non Seasonal	Enable this check box if the data associated with this causal factor is not known to be a predictable function of time. For example, price varies with time, but randomly, so price should be flagged as non seasonal. See "Causal Factors and Engine Models".
Self Shape Indicator	Enable this check box if this promotion causal factor should be represented as a shape. See "About Promotion Shape Modeling".
IG Shape Indicator	Enable this check box if this promotion causal factor should be represented via shape modeling when it is aggregated to the IGL.
Omit Seasonal	Enable this check box if you want to nullify values of the global seasonal causal factors for the time buckets during which the causal factor occurs. Specifically this refers to the causal factors d1—d12 or d1—d7 and m1—m12. For example, if you have monthly data and you omit seasonal effects for a given causal factor Promo1, that means that Demantra switches off the causal factors for the duration of Promo1. By omitting seasonal effects, you enable Demantra to capture the promotion shape more clearly for the shape analysis. This option is suitable only if you expect the effect of this causal factor to be much stronger than the seasonal effects. If causal factors overlap each other, then Demantra gives precedence to the causal factor that you have flagged to omit seasonal effects.
Num Shapes	Specify the maximum number of allowed shape causal factors for the engine to use for a given node in the forecast tree, for this promotional causal factor. Use an integer from 0 to 8, inclusive.

6. Click Validate to check the validity of the configuration.
7. Click Save.
8. Now you can return to previous dialog boxes to make further changes. Or click Finish to exit.

Adjusting the Promotion Dates

By default, Promotion Effectiveness assumes that a promotion has an effect between its start and end dates, as provided to Oracle Demantra. Typically the promotion has an actual effect in a slightly different span of time, as in the following example:



You can adjust the dates used by the Analytical Engine in two complementary ways:

- To move the end date of the promotion, set ShiftDynPromoDate to a different end date.

  Note: Alternatively, to specify an overall shift in time for all promotions, set the ShiftPromoCausals parameter.

- To stretch a promotion by adding time buckets to the beginning or end, do the following.
  - Decide which attribute or attributes have pre and post-promotional effects.
  - Enable shape modeling for those promotional causal factors; see "About Promotion Shape Modeling".
  - For those promotional causal factors, set Num Shapes equal to 1.
  - The user must change the pre_effect and post_effect settings of the combination, which default to zero. These settings (in mdp_matrix) specify the number of buckets to search backwards and forwards outside the promotion dates. In the preceding example, we set pre_effect equal to 2.

Typically you also set the ShiftPromoMaxValue parameter, to make sure that you adjust the dates of promotions in the near future (rather than adjusting only historical promotions).

See also

"How the Analytical Engine Uses Promotions"

"Configuring Global and Local Causal Factors"

"Deleting a Causal Factor"

About Promotion Shape Modeling

In shape modeling, you capture the profile of the demand over the duration of a promotion. The Analytical Engine models the overall demand as a linear combination of Oracle proprietary shapes, as many as eight shapes; this information replaces the normal causal factor that would have been used instead. The Analytical Engine automatically associates a different shape with each value of the promotional attribute that uses shape modeling. The Analytical Engine calculates the coefficients for each shape, for each relevant combination.

> **Note:** The feature described here is available *in addition to* activity shape modeling; see "About Activity Shape Modeling".

When to Enable Shape Modeling

You should enable shape modeling only if the following are all true:

- The demand data is continuous (point-of-sale data rather than order data).
- The typical length of a promotion is more than one time bucket.
- You need to search for pre and post effects of promotions.

Other Considerations

Remember that when you enable shape modeling for a promotional causal factor, the single promotion causal factor is replaced by up to eight causal factors. If the promotional causal factor is transposed, that adds even more causal factors: up to eight for each column that the transpose creates.

To keep the number of causal factors down, you can specify the maximum number of shapes permitted for any given promotional causal factor.

Enabling Promotion Shape Modeling

To enable promotion shape modeling:

1. Identify the promotional causal factors that you want to represent as shapes.
2. On the Promotional Causal Factors screen, make sure to check the Self Shape Indicator option for each of those promotional causal factors.
3. Consider also setting the following options.

- IG Shape Indicator
- Omit Seasonal
- Num Shapes

7

Tuning the Analytical Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

It is usually necessary to adjust some parameters to configure the Analytical Engine correctly before running it the first time. Other adjustments can be made later to optimize the behavior and performance.

This chapter covers the following topics:

- Editing Engine Parameters
- Creating or Renaming Engine Profiles
- Tuning Analytics
- Tuning Performance
- Reconfiguring the sales_data_engine Table
- Enabling Engine Models Globally
- Configuring the Engine Mode
- Advanced Analytics (Nodal Tuning)
- Forecast Tree Check

Editing Engine Parameters

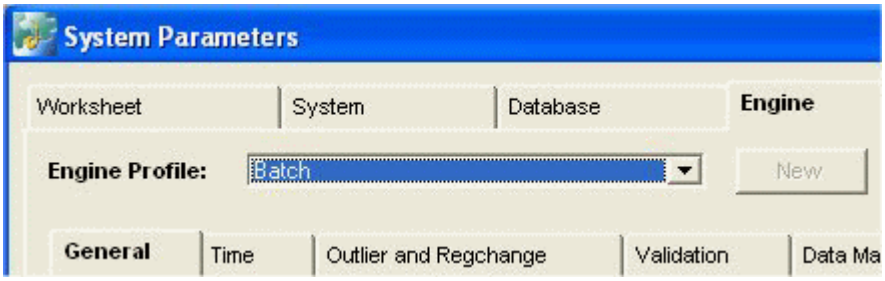
To tune the Analytical Engine, you modify values of two types of engine parameters:

- Global parameters that apply to the engine or to most or all of the forecasting models. For convenience, you define *engine profiles* , which are sets of engine parameters with specific values. Demantra provides some predefined profiles for different purposes, and you can define additional engine profiles, as needed.

- Parameters that apply to specific forecast models.

To edit the global engine parameters:

1. Log onto the Business Modeler.
2. Click Parameters > System Parameters. The System Parameters dialog box appears.
3. Click the Engine tab.



4. From the Engine Profile drop-down, select the engine profile whose parameter settings you want to adjust.
5. Find the parameter of interest. The dialog box provides find, sort, and filter capabilities to help you with this. See "Engine Parameters".
6. To change the value of the parameter, click the Value field for that parameter. Type the new value or select a value from the drop-down menu.
7. Click Save to save your changes to this profile.
8. Click Close.

To edit specific model parameters:

To edit most model-specific parameters, you must work directly within the Demantra database. For information on the parameters and their locations in the database, see "Engine Parameters".

See also

"Creating or Renaming Engine Profiles"

"Tuning Analytics"

"Enabling Engine Models Globally"

Creating or Renaming Engine Profiles

Engine Profiles and Parameter Inheritance

When the engine is run with a specific engine profile, it references any parameter settings associated with that profile. Profile settings that are not found are inherited from other profiles.

For Batch engine profiles, any parameters that are not found in the profile will receive values from the Base engine profile. For Simulation and Subset engine profiles, any parameters that are not found in the profile are referenced in that parameter's parent batch profile. If the parameter is found in the parent batch profile, then that value will be used. Otherwise, the values from the Base engine profile are used.

The same logic is applied when displaying engine parameters in the Business Modeler. If a parameter exists in for the specified profile, its value is shown. If the parameter is not found then values based on inheritance are displayed instead.

To create or rename an engine profile:

1. Log onto the Business Modeler.
2. Click Parameters > System Parameters. The System Parameters dialog box appears.
3. Click the Engine tab.
4. Do one of the following:
   - To rename an existing profile, click the profile from the Engine Profiles list and then click Edit.
   - To create a new profile, click New.
5. Enter a (new) name for the profile.

   See also

   "Editing Engine Parameters"

   "Tuning Analytics"
6. When creating a new Engine Profile, determine whether it is to be a batch or a simulation profile. A simulation profile must be attached to a parent batch forecast, because the simulation is stored in the sim_val column matching the batch parent profile.

   Example
   For example, a simulation with the batch parent profile ID of 3 is stored in the sim_val_3 column. The internal profile ID can be found in ENGINE_PROFILES

table.

7. If this is a simulation profile, select the Simulation Engine Profile check box.

8. If the new profile is a simulation profile, select the Parent batch Profile from the drop-down menu.

9. Click OK.

Tuning Analytics

For basic parameters related to the forecast tree, see "Specifying Additional Parameters". For information on all parameters (including default values), see "Engine Parameters".

Analytical Parameters

The following parameters control analytics:

Parameter	Purpose
UseNonNegRegr	Specifies whether to allow negative coefficients. Most of the models use this parameter. In cases with multiple, possibly co-varying causal factors, the Analytical Engine sometimes finds a solution that includes a large positive coefficient for one causal and a large negative coefficient for another causal factor, so that they nearly cancel one another. Mathematically, this solution may be good. But a negative coefficient means that the demand acts in the opposite sense to the causal factor; that is, demand drops when the causal factor increases. And a negative coefficient does not make sense in the vast majority of cases. This means that it is generally good practice to disable negative coefficients.
ShapeSign	Specifies the signs for the shape causal factors when using them in non negative regression.
NumShapes	Specifies the maximum number of allowed shape causal factors for the engine to use for a given node in the forecast tree. Use an integer from 0 to 8, inclusive. This applies to activity shape modeling (rather than to promotional shape modeling).

Parameter	Purpose
CannibalizationIgnore	Controls whether the Analytical Engine will calculate switching effects (cannibalization). You can use this parameter to switch off that calculation in order to check that the Analytical Engine is calculating the basic lift appropriately.

Parameters Related to Promotional Causal Factors (PE Mode Only)

The following parameters are related to promotional causal factors:

Parameter	Purpose
PromotionStartDate	Earliest date for which promotion data can be considered reliable.
ShiftDynPromoDate	A date that overrides the default end_date column from the promotion date, thereby shifting the end date of the promotion. It can be an sql expression that returns a date value. Alternatively, to specify an overall shift in time for all promotions, set the ShiftPromoCausals parameter.

See "Adjusting the Promotion Dates".

Parameters Related to Validation (PE Mode Only)

The Analytical Engine applies different forecasting models to each node of the forecast tree, calculates the uplift for each node, and uses that uplift to check whether the model is appropriate for that node. If not, the model is not used for the node.

The Analytical Engine can discard a model for a given node for either of two reasons:

- The model generated an uplift that was beyond the upper allowed bound, as specified by the UpperUpliftBound parameter.
- The model generated too many exceptional uplifts. An uplift is considered "exceptional" if it exceeds the lower bound specified by the LowerUpliftBound parameter. The AllowableExceptions parameter controls how many exceptional uplifts are permitted.

Parameters Related to Output (PE Mode Only)

The following parameters control the output of Promotion Effectiveness forecast values:

Parameter	Purpose
NormalizeResults	Specifies whether to normalize the historical engine results so that the observed baseline values are preserved. If you normalize the engine results, note that the Analytical Engine writes these results to different fields in promotion_data than it does otherwise. See "Key Tables".
WriteMissingDatesUplift	Specifies whether to write uplifts for dates that are missing from sales_data. If you specify no, then the Analytical Engine writes uplifts only for dates that already have sales. However, the uplifts will not necessarily add up to the total uplift.
UpliftThresholdValue	Specifies a threshold for uplift values. If the Analytical Engine calculates uplift values below this threshold, those values are dropped rather than being written to the database.
UpliftThresholdMethod	Specifies whether the previous threshold is expressed as an absolute value or as a percentage of baseline.

See also

"Editing Engine Parameters"

"Creating or Renaming Engine Profiles"

Tuning Performance

To improve the performance of the Analytical Engine, check the settings of the following parameters. To access these parameters in Business Modeler, click Parameters > System Parameters and then click the Database tab.

Basic Engine Parameters for Performance

The following engine parameters are critical to good performance. Make sure they are set appropriately for your configuration.

Parameter	Purpose
min_fore_level	Minimum forecast level that the engine will forecast. For PE, this must be at or above the lowest promotional level (LPL). Make sure this is defined appropriately for your forecast tree.
start_new_run	Specifies whether to start a new Analytical Engine run or to perform an engine recovery. Use yes or prompt.
node_forecast_details	Specifies whether the Analytical Engine should write forecast data for each node (the NODE_FORECAST table), before splitting to lower levels. Writing to this table slows the engine, so you should switch off this option unless you have tested that the impact is acceptable.
WriteIntermediateResults	Specifies whether to enable the advanced analytics function, which is available only on the desktop. Make sure this option is off unless you have tested that it does not interfere unduly with performance.
BulkLoaderBlockSize	Specifies the minimum number of rows that Analytical Engine loads at one time, when writing to the database. The larger this is, the more quickly the data is loaded, but there is greater risk if the database connection is lost. For a high-volume system, use 20,000.
BulkLoaderEnableRecovery	Specifies whether Oracle Bulk Loader should perform recovery after a lost database connection. For a high-volume system, use 0.

Parameters That Can Speed Performance

The following parameters can help the Analytical Engine run more quickly by omitting processing steps. You should change these only if you are sure that doing so will not cause problems.

Parameter	Purpose
ForecastGenerationHorizon	Specifies what historical fit data the engine will write to the database. If this parameter is 0, the engine writes the forecast only. If this parameter is a positive integer N, the engine writes the last N historical fit values.
ResetForeVals	Controls the method of clearing current forecast values for the forecast version currently being generated. If set to Yes (default), then all combinations with prediction status of 97, 98, or 99, fore = 2, will get null forecast values and active combinations will be overwritten by the new forecast. If set to No, then the existing forecast for inactive combinations will not be cleared. If set to All, then all combinations will have their forecast cleared regardless of prediction status. Note that setting this parameter to 'All' may substantially increase engine run time. If you want to reset the forecast outside the engine date range, set this option to All. The individual engines perform this function during the run. Each engine produces a list of the inactive nodes for the branch/simulation it is processing and adds special rows for the bulk loader. Procedures ProcessTempSaleTable and ProcessTempPromoTable perform the resetting. They update the configured data table and the PROMTION_DATA table. Engine parameter DBHintInitialForeClean applies only to the functionality of Engine Manager parameter DeleteIsSelfRows. If set to All, then besides resetting all inactive combinations, the engine will reset all non-updated rows during the current run for active combinations as well. The procedure detects which rows were not updated according to the LAST_UPDATE_DATE column. The option "All" cannot be used with EngineOutputThreshold>0 in the same engine profile.
RunInsertUnits	Specifies whether the Analytical Engine calls the INSERT_UNITS procedure at the start of an engine run. This procedure makes sure the engine has rows to write into when generating the forecast. For information on all procedures, see "Database Procedures".

Parameter	Purpose
BatchRunMode	**Applies to PE mode, and applies to both batch run and simulation run.** Specifies the kind of forecasting to do: • 0=run the forecast against only the learning (estimation) • 1=run the promotion forecast (the normal setting) • 2=perform an estimation and promotion forecast run (fast simulation; this option uses previously cached data) For options 0 and 2, the Analytical Engine performs fewer scans. (For details on the engine flow, see "Promotion Effectiveness Engine Phases".)
align_sales_data_levels_in_loading	Specifies whether to adjust the sales_data table for direct use by the engine (instead of the sales_data_engine table). • 0=no (do not adjust the sales_data table for direct use by the engine) • 1=yes (adjust the sales_data table) For information on this parameter, see "Non-Engine Parameters".
start_date	Beginning of historical data used by the engine. Used together with parameter HistoryLength. If left at default 01/01/1995, may require the engine to find first period of real history in historical demand. For larger environments this can add significant time to the engine run. It is strongly recommended this parameter be reset to beginning actual date where history begins.

The engine divider uses Fast Divider functionality. The engine uses the ENGINE_BRANCH_LIST table to determine the actual branch, not the BRANCH_ID column of the configured combination table. Each time the engine processes a branch, it updates the BRANCH_ID column with the actual allocation.

Database Partitioning for the Engine

You can partition the database so that the Analytical Engine can access data more rapidly. Specifically, you can place different parts of the sales_data, mdp_matrix, and promotion_data tables on different partitions, so that each partition corresponds to a

potentially different item and/or location.

The overall process is as follows:

1. Create the partitions and move rows to them as needed. This is beyond the scope of this documentation.

2. To partition only by item, choose a database column that you can use to subdivide the records by item. This column must exist in the sales_data, mdp_matrix, and (in the case of Promotion Effectiveness) promotion_data tables and must have the same name in each of these tables.

   For example, it might be suitable to partition by brand. The brand information is available in mdp_matrix as (for example) the t_ep_p2a_ep_id field. You would have to replicate this column to the sales_data and promotion_data tables as well, perhaps by a database trigger.

   Similarly, to partition only by location, choose a database column that you can use to subdivide the records by location.

   To partition by item and by location, choose a database column that you can use to subdivide the records by item and another column that subdivides them by location.

3. Set the following parameters so that the Analytical Engine can find the partition on which any combination resides:

Parameter	Purpose
PartitionColumnItem	Specifies the name of the column that partitions the data by item.
PartitionColumnLoc	Specifies the name of the column that partitions the data by location.

Other Database Considerations

Pay attention to the indexes of sales_data and mdp_matrix tables.

Also, for Oracle databases, Demantra writes to multiple tablespaces, as specified during installation. The tablespace assignments are controlled by parameters, which you can edit through the Business Modeler. Make sure that these parameters refer to tablespaces within the appropriate database user, and make sure each has enough storage.

> **Note:** Oracle recommends that you use the standard names for these tablespaces, as documented in the Oracle Demantra Installation Guide.

Then it is easier for you to share your database with Oracle Support Services in case of problems.

Additional parameters control the default initial sizes and how much storage is added.

Parameter	Description
initial_param	Default initial size of system tablespaces.
next_param	Incremental amount of storage that is added to a tablespace when more space is needed.
tablespace	Tablespace used for the sales table.
indexspace	Database index space that stores the forecast table indexes, as specified during installation.
simulationspace	Tablespace used for simulation data.
simulationindexspace	Tablespace used for simulation index data.
sales_data_engine_index_space	Tablespace used for the index of sales_data_engine.
sales_data_engine_space	Tablespace used for sales_data_engine table.

*For information on these parameters, see "Non-Engine Parameters".

Reconfiguring the sales_data_engine Table

The Analytical Engine creates and uses a table (or view) called sales_data_engine. You can control how the Analytical Engine does this, in order to improve performance.

- You can adjust the sales_data table for direct use by the Analytical Engine, so that the sales_data_engine table is not needed.
- Normally, the Analytical Engine internally creates the sales_data_engine table for its own use, and creating this table can be time-consuming. You can speed up the engine by configuring it to use the sales_data table instead of the sales_data_engine table.
- Normally, when the Analytical Engine runs, it joins sales_data_engine (or its synonym) with the mdp_matrix table. This is not always necessary, and you can prevent this join to speed up the Analytical Engine.

The following table lists the key parameters and some typical settings:

Parameter	Description	Normal batch run	Normal simulation	Faster engine run*	Fast simulation*
align_sales_data_levels_in_loading**	Specifies whether to adjust the sales_data table for direct use by the engine (instead of the sales_data_engine table). • 0=no (do not adjust sales_data) • 1=yes	0	0	1	1
SdeCreateSwitch	Specifies the type of logic to use in order to create the sales_data_engine table. • 0=Use internal engine logic • 1=Use external logic, as specified by stored @ procedures, create_process_temp_table, create_object and drop_object @ procedures. These procedures may be modified by consultants.	0	0	1	1

Parameter	Description	Normal batch run	Normal simulation	Faster engine run*	Fast simulation*
SdeAnalyzeSwitch	Specifies the type of logic to use in order to analyze the sales_data_engine table. • 0=no. The engine assumes analyze is already performed and creates indexes as part of external procedures. • 1=yes. The engine analyzes and creates indexes. This is the default value.	1	1	0	0
SdeCreateJoin	Specifies whether the Analytical Engine should join sales_data_engine (or its synonym) and mdp_matrix during its run. • 0=no (do not join these tables) • 1=yes (join these tables; this is the default)	0	0	0	1

*See "Additional Steps". Also note that fast simulation forecasts future uplift only. **For information on this parameter, see "Non-Engine Parameters".

Additional Steps:

1. Configure the forecast tree as you normally would. See "Configuring the Forecast Tree".

2. In the database, create a synonym for sales_data. The name of synonym should be sales_data_engine or whatever synonym you plan to use.

3. Rewrite the following database procedures as needed:
   - create_process_temp_table
   - create_object
   - drop_object
4. Consult Demantra for assistance.
5. Test that you have configured the engine correctly.
   1. Add new records to sales_data, in any of the following ways: by loading via the Data Model Wizard, by running integration, or by chaining.
   2. Run the engine.
   3. Check the sales_data table and make sure of the following:
      - This table should have a column for every level in the forecast tree,
      - This table should have a column named do_aggri.
      - This table should have should include non-null data in these columns for at least some of the records.

Enabling Engine Models Globally

Demantra provides a set of theoretical engine models that the Analytical Engine uses when it creates a forecast. Usually you do not make changes, but you can specify which models to use, as well as set basic parameters for each model.

Caution: Only advanced users should make these changes.

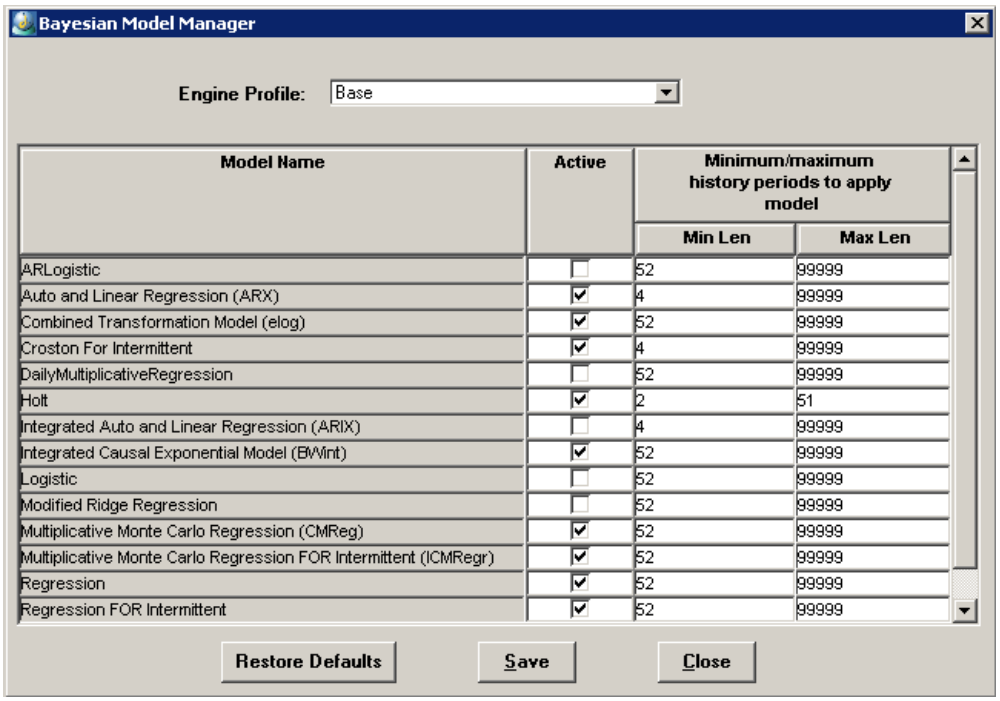
When the Analytical Engine runs, it may use a subset of these models on any particular combination. The engine tests each model for applicability; see "The Forecasting Process".

Note: Optimization can only use linear generalized coefficients

To enable models for the Analytical Engine to use:

1. Log onto the Business Modeler.
2. Click Engine > Model Library.

The following dialog box appears.



3. Select the batch engine profile to be associated with the model library configurations.

4. For each model, do the following:

   - To enable the Analytical Engine to use this model, make sure the Active check box is checked. For details on these models, see "Theoretical Engine Models". Note that not all models are supported with any given Analytical Engine.

   - The other two settings control the minimum and maximum number of non zero observations that a combination must have in order for the Analytical Engine to consider using this model for this combination. To specify these values, type integers into the Min Len and Max Len fields.

     Note: Min Len must be equal to or greater than the number of causal factors in the forecast, except for the HOLT and FCROST models, which do not use causal factors.

5. Click Save and then click Close.

Configuring the Engine Mode

Oracle provides two different modes of the Analytical Engine:

- The DP mode is for use with Demand Planner or other planning products.
- The PE mode is for use with Promotion Effectiveness.

To specify the engine mode

The RUNMODE parameter specifies the mode of the Analytical Engine to use:

- Use 1 to specify the PE mode.
- Use 0 to specify the DP mode.

If you use this setting, also be sure that you have defined the forecast tree appropriately. In particular, make sure that the LPL (PROMO_AGGR_LEVEL) is the same as the minimum forecast level. To set this, use the forecast tree editor in the Business Modeler.

See also

"Troubleshooting"

Advanced Analytics (Nodal Tuning)

Normally, the Analytical Engine uses the same options for every node in the forecast tree, but you can make certain adjustments for individual nodes, if necessary. This task is recommended only for advanced users in conjunction with Oracle Support.

Of the models you specify for a given node, when the Analytical Engine runs, it may use a subset of these models, as described in "The Forecasting Process". The Analytical Engine indicates (in the models column of mdp_matrix) the models that it used.

To enable advanced analytics :

1. Set the usemodelspernode and UseParamsPerNode parameters to yes.
2. Then for each node of the forecast tree, you can specify engine models and engine parameters for different nodes in the forecast tree. To do so, you use the Analytics window.

Forecast Tree Check

The Forecast Tree Check process ensures that the engine tree configuration and its levels are valid. The forecast tree levels and additional definitions are defined in the Business Modeler. The Forecast Tree Check verifies that each node has only one parent

node in the engine level hierarchy, essentially ensuring a clean parent child relationship between all levels participating in the forecasting process.

The engine enforces this hierarchy to ensure that, when dividing the overall batch process into smaller tasks, a node will not be forecasted by two different tasks when going up levels in the forecast tree. This avoids database server deadlocks when more than one engine tries to update the same combination and ensures consistent results as each forecast node receives its forecast from a single source.

You run the Forecast Tree Check from the Engine Manager by specifying a Run Mode of '10' (Tree Check). Other run modes that the Engine Manager supports include 1 (Batch) and 99 (Simulation).

The Forecast Tree Check process is run from the command line.

8

Using the Engine Administrator and Running the Engine

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

Before you run the Analytical Engine for the first time, it is useful to ensure that you have configured it correctly. This chapter describes how to administer the Analytical Engine.

This chapter covers the following topics:

- Before Running the Analytical Engine
- General Notes about Running the Analytical Engine
- Deploying the Analytical Engine
- Configuring Engine Settings
- Deploying Demantra CDP RAC Services
- Running the Analytical Engine from the Start Menu
- Running the Analytical Engine from the Command Line
- Running the Analytical Engine from a Workflow
- Stopping an Analytical Engine Run
- Running the Simulation Engine
- Running Engine Starter
- Troubleshooting
- Oracle Wallet Troubleshooting
- Viewing the Engine Log
- Examining Engine Results

- Running the Engine in Recovery Mode
- Stopping the Engine

Before Running the Analytical Engine

Before you run the Analytical Engine for the first time, it is useful to ensure that you have configured it correctly.

- Make sure that you have installed the correct version of the Analytical Engine and that you have set the RUNMODE parameter correctly; see "Configuring the Engine Mode".

- Make sure the engine is deployed on all the machines where you want to use it. See "Deploying the Analytical Engine."

- Make sure that you have enough (and not too many) observations for every node in your forecast tree, as needed by the engine models you plan to use.

  If a node is left with no suitable model, the Analytical Engine will not forecast on that node. Instead it will forecast at a higher level, if possible.

- Various configurable fields contain parts of SQL queries used by the engine during run and may fail the engine if configured incorrectly. Common reasons for failures are misspellings, references to non existent columns, or using functions or syntax not compatible with the database server.

  To check all your engine-related SQL, check the following tables:

  - In the Init_Params_* tables, check the parameters quantity_form and UpTime.

    The default expression for quantity_form transforms negative values to zero. This is considered best practice, and should be modified only if there is a direct need for the Analytical Engine to see negative demand and for negative proportions to be calculated. Care must be taken when enabling negative proportions, as allocation using negative proportions may result in violation of business rules.

  - In the causal_factors table, the Local_Funct column uses SQL.

  - (For PE mode) In the m3_causal_factors table, many settings here use SQL.

- Make sure that the database is configured correctly, specifically the table extents. Also, if you have loaded the Demantra schema from a dump file, make sure that the current database contains table spaces with the same names as in the original database.

General Notes about Running the Analytical Engine

- The first engine run takes longer than later runs. This is because the Analytical Engine must set up internal tables on its initial run. You can reduce the length of time of the first engine run; see "Reconfiguring the sales_data_engine Table".
- You cannot run the Analytical Engine in batch mode unless the Business Logic Engine is closed and no simulation is running.

Also see

"Introduction to the Analytical Engine"

Deploying the Analytical Engine

The installer deploys the Analytical Engine for you, but in case of problems, you can deploy the engine manually. To do so, run the batch file Demantra_root/Demand Planner/Analytical Engines/bin\RegEngine.bat.

For information on deploying the Analytical Engine, refer to *"Deploying the Demantra Analytical Engine"* in the Demantra Installation guide.

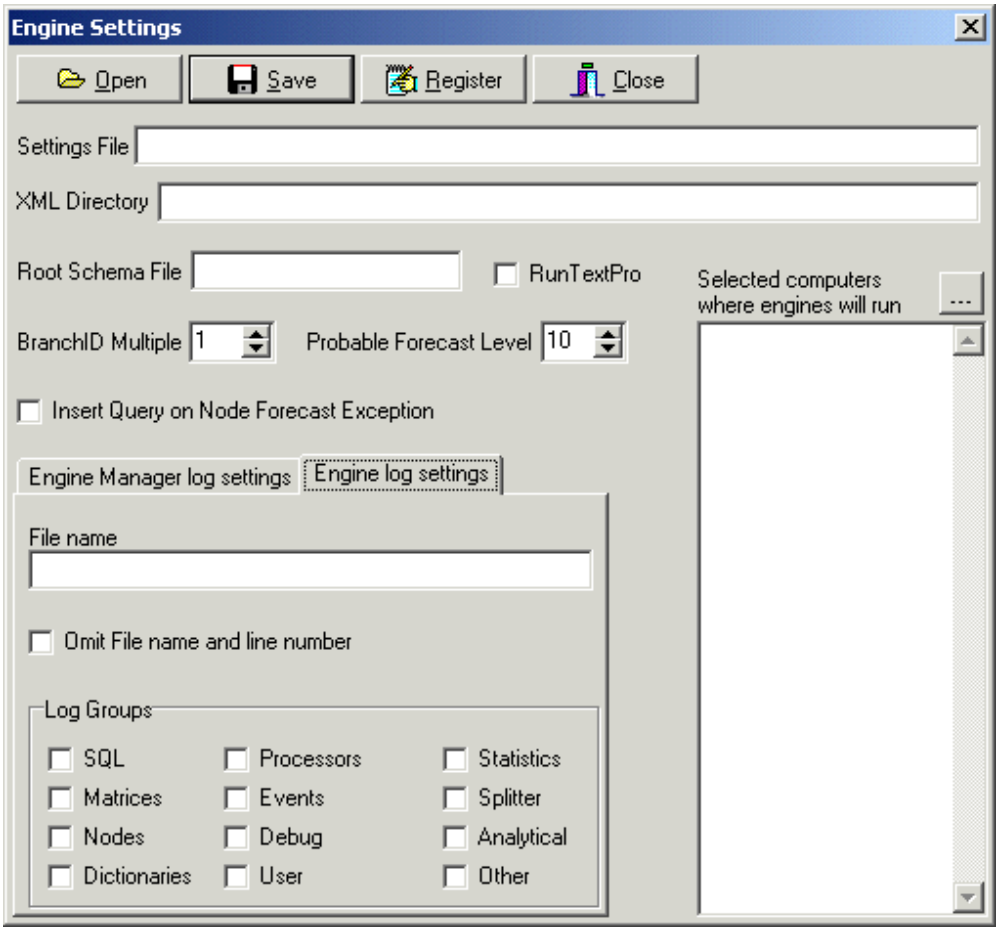
Configuring Engine Settings

Engine configuration settings are edited in the Engine Settings window and saved in the file named Settings.xml. When the engine starts, it reads the settings from this file. Settings.xml can also be edited manually using an XML editor.

To open the Engine Settings window:

1. Start the Engine Administrator.
2. Click Settings > Configure Engine Settings. Or click the Configure Engine Settings button.

   The Engine Settings window appears.



To load settings:

1. Click Open.
2. Select Settings.xml from the bin directory of the Analytical Engine.

   The Settings File field displays the location of Settings.xml.
3. Complete the fields as needed; see "Engine Settings".
4. To save your settings, click Save.
5. To register your settings, click Register.

Engine Settings:

You can configure the following settings.

Setting	Meaning
Selected computers where engines will run	The Engine Manager tries to create and initialize all the Engines specified in this list. You can choose one or more machines on which the Analytical Engine has been installed. These machines must also be running the appropriate database client software, so that they can communicate with the Demantra database. In order to run the Analytical Engine on multiple machines, your system must include the Distributed Engine.
Engine Manager Log Settings	
File name	Path and filename of the log file that will record errors from the Engine Manager.
Output Target	Select either stdout or File. If you choose stdout, the output is sent to the Log File window. In this case, you can still save the log to a file, from the Run Engine window.
Log Groups	Specifies what level of details to log for a specific Log Group.
Omit file name and line number	Select this option if you do not want the engine log file to include this information.
Omit Time	Select this option if you do not want the engine log file to include this information.
Engine Log Settings	
File name	Path and filename of the log file that will record errors from the engine itself.
Analytical Log Group	This field should be left blank during standard engine runs. If you encounter a problem when running the engine, you may be instructed by Oracle Support to enter a specific value in this field. This will enable a greater level of detail in the engine log file, but should be done only when troubleshooting engine issues.
Omit file name and line number	Select this option if you do not want the engine log file to include this information.

Setting	Meaning
Omit Time	Select this option if you do not want the engine log file to include this information.

Deploying Demantra CDP RAC Services

The Demantra CDP RAC Services feature allows the system to override the default RAC load balancing method to improve performance of the Analytical Engine in a RAC environment. This enhancement is available only when using the Oracle Demantra Consumption-Driven Planning (CDP) module, when running Real Application Clusters (RAC) on one of Oracle's Engineered Systems platforms (for example, Exadata). Additionally, this feature is currently available only in the 64-bit version of the Analytical Engine and the cluster-enabled version of the Batch Logic Engine (BLE), which runs with the 64-bit version of the Analytical Engine.

By default, Analytical Engine connection requests to the RAC services are dispatched in a "round robin" fashion. That is, each connection is established as needed and none of the connections has priority over another. Since the Analytical Engine creates more than one connection, each connection could be handled by a different RAC service, which can increase network traffic between the different RAC nodes and affect overall performance. By using the CDP RAC services feature, all of the connections created by a specific engine task can use the same RAC service, thereby decreasing network traffic and improving system performance.

For information on enabling the CDP RAC Services, refer to *"Real Application Clusters (RAC) Advanced Setup with Oracle Wallet"* in the Oracle Demantra Installation Guide.

Running the Analytical Engine from the Start Menu

For information on deploying the Analytical Engine, see "Deploying the Demantra Analytical Engine" in the *Demantra Installation Guide*.

To run the Analytical Engine from the Start menu:

1. To run the engine in batch mode: on the Start menu, click Programs. Then click Demantra > Demantra Spectrum release > Analytical Engine.

   See also

   "Running the Engine from the Command Line"

Running the Analytical Engine from the Command Line

Running the Analytical Engine with Oracle Wallet Defined:

1. Run the EngineStarter script once for each host machine, either as background process or in a different shell/command console. This must be repeated if EngineStarter was killed, the shell from which it was run was aborted or the host on which it was running was restarted.

   Note: Oracle recommends that you configure EngineStarter as a service that runs automatically whenever a machine is restarted.

   For example:

   ```
   ./EngineStarter.sh &
   ```

   Engine Starter will register itself as active in the Database and listen for requests.

2. Run the Start_Engine2K script with a profile ID as a parameter to start a batch run.

   For example:

   ```
   ./Start_Engine2K.sh [Profile ID]
   ```

   Engine Manager searches the database for active Engine Starters and uses them to spawn Analytical Engine processes as needed.

Running the Analytical Engine without a Defined Oracle Wallet:

1. Run the EngineStarter script once for each host machine, either as background process or in a different shell/command console with the database connection details.

   For example:

   ```
   ./EngineStarter.sh [DB host address]:[port]/[service name] [user]
   [password] &
   ```

   Engine Starter will register itself as active in the Database and listen for requests.

2. Run the "Start_Engine2K" script with a profile ID as parameter to start a batch run.

   For example:

   ```
   ./Start_Engine2K.sh [Profile Id] [DB host address]:[port]/[service
   name]  [user]  [password]
   ```

   Warning: This is not a secure method for running the Analytical Engine and should be avoided.

Running the Analytical Engine from a Workflow

1. EngineStarter must be running as background process before the Demantra application and workflow are started. This can be done either manually from a console, or automated via startup scripts of the host operating system.

2. Engine Manager must be physically located on the same host as the Demantra Application Server.

3. Make sure the SYS_PARAMS.EngineBasePath parameter is set correctly, pointing to the Engine Root Directory where all scripts are located.

4. You must use Engine scripts to either start or stop an Analytical Engine run. Either **Start_Simulation2K**, **Start_Engine2K** or **KillEngine** should be used from within workflow steps. Use the appropriate script extension and foreword/backward slash for the host operating syswtem.Usually the parent workflow step implements a condition step to determine the type of operating system and branch to the appropriate execution step.

5. Any custom or existing executable step calling one of the Engine scripts must be preceded with the new path token, pointing to where Engine scripts are located, rather than a hardcoded path or the old token pointing to the Demantra application base path.

   For example:

   ```
   #EngineBasePath#\Start_Simulation2K.bat
   ```

Stopping an Analytical Engine Run

Run the KillEngine script to stop an existing run. For example when Engine is running in simulation mode and user wants to start a batch run. Engine Starter will not be killed.

Running the Simulation Engine

1. Make sure that Engine Starter is running.

2. Run the Start_Simulation2K script to start the simulation Engine either with or without database connection details (depending on the setup method).

   For Example:

   ```
   ./Start_Simulation2K.sh
   ./Start_Simulation2K.sh [DB host address]:[port]/[service name]
   [user]  [password]
   ```

Running Engine Starter

Engine Starter should be executed once on each engine host.

Engine Starter script can handle 0, 3, 4 or 5 command line parameters. Any other number will display the error and usage string.

If you run Engine Starter with one argument you will get the usage details. For example:

```
EngineStarter.bat [parameter]
```

To start Engine Starter use the following commend:

```
EngineStarter.bat [ mandatory CONNSTR ] [ mandatory USERNAME ] [
mandatory PASSWORD ] [ optional STARTER_ID ] [ optional ENGINES_NUM ]
```

Examples:

```
EngineStarter.bat
EngineStarter.bat  [DB host address]:[port]/[service name]  [user]
[password]
EngineStarter.bat  [DB host address]:[port]/[service name]  [user]
[password] [Starter ID] [Num Engines]
EngineStarter.bat  [DB host address]:[port]/[service name]  [user]
[password] [Starter ID] [Num Engines]
```

Possible Failures:

1. Cannot connect to the database:

   Solution: Check Oracle wallet setup or provided connection details.

2. Failed to create a queue for the chosen starter ID.

   Solution: Check that chosen ID is not over eight characters, as it is appended/prepended with some additional chars and should eventually be a valid table name. Check that no invalid characters are part of the chosen ID. Try manually setting the starter ID in the setenv script to something short and simple like "starter1" or directly pass this ID via command line.

Troubleshooting

This section contains tips that address specific error conditions that you could encounter:

- If the Analytical Engine fails to run, see the list in "Before Running the Analytical Engine".

- If the engine failed while running an SQL statement, check the following logs:
  - manager.log

- engine2k.log

  Find the offending SQL and try running it within a third-party database tool to identify the problem.

  If the engine iterator failed, resulting in the error "node not found in map," that indicates a problem in the mdp_matrix table. Usually, this means that you need to set the align_sales_data_levels_in_loading parameter to true and then run the MDP_ADD procedure. (For information on this parameter, see "Non-Engine Parameters".)

- If the Analytical Engine run does not finish and gives a message saying that it is stacked at some node or that it "does not have a usable number of observations," this means that the mdp_matrix table is not in a good state. To correct the problem, run the MDP_ADD procedure.

- If the Engine Log displays the message "Can not initialize caches" that may mean that your database is too large for the given number of branches. Reconfigure the engine to run on more branches and try running it again.

- If the Analytical Engine fails or generates errors when processing large amounts of data, make sure the MaxEngMemory parameter is set to zero.

- **PE only:** If you receive a message like "ERROR Node not found in map", that means that something is wrong with synchronization between sales_data and mdp_matrix. To correct the problem, truncate mdp_matrix and run the MDP_ADD procedure.

- If the Analytical Engine takes an unreasonably long amount of time to create the sales_data_engine or the promotion_data table, make sure that you have done an analyze table on these tables.

- If you receive a message such as "Description: ORA-00959: tablespace 'TS_SALES_DATA' does not exist," that typically means the dump file you installed refers to different table spaces than you have in the current database. Reassign the Demantra table spaces by changing the parameters that control them:

  - indexspace
  - sales_data_engine_index_space
  - sales_data_engine_space
  - simulationindexspace
  - simulationspace
  - tablespace

For information on these parameters, see "Non-Engine Parameters".

Validating Input Parameters

Validating engine and model input parameters is used to identify the source of errors caused by configuration issues and errors. This streamlines and shortens the troubleshooting process and reduces the need for support.

- **Parameters:**
  1. The Analytical Engine loads the 'Parameters' data from the PARAMETERS table.
  2. The engine then loads the 'Parameters' data from 'Parameters Daily.xml', 'Parameters Monthly.xml', or 'Parameters Weekly.xml' depending on 'timeunit'.

- **InitParams:**
  1. The Analytical Engine loads the 'InitParams' data from INIT_PARAM_0 table.
  2. Then the engine loads the 'InitParameters' data from 'Init Params 0 Daily.xml', 'Init Params 0 Monthly.xml', or 'Init Params 0 Weekly.xml' values.

The Analytical Engine loops through parameters from xml, validates them against the database parameters, fixes the collected parameters, or adds the missing parameters in the database.

The validation rules are configurable. If they belong to the current run, you can specify the parameter group, and the restrictions by which the parameters are compared.

If any of the input parameters fails the validation, the system replaces the erroneous parameters with the default value if the restriction does not contain '?'. Otherwise, the system simply generates a warning message to inform the user of the erroneous input parameter.

> **Note:** Demantra supports only the following type "double" validations for parameters:
>
> - 1 - All the groups - always validate
> - 2 - DP batch
> - 3 - PE batch
> - 4 - DP simulation
> - 5 - PE simulation

Example 1

```
<Entry>
  <Key argument="AllowNegative"/>
  <Value type="double" argument="0"/>
  <Validate group="1" restrict="=1,=0"/>
</Entry>
```

The above-mentioned validation means "Allow Negative" parameter of type "double" with default value "0". The validation belongs to group "1" thereby run during all engine runs and its value can either be "1" or "0"

Example 2

```
<Entry>
  <Key argument="lead"/>
  <Value type="double" argument="52"/>
  <Validate group="3,4" restrict="&gt;0,?&lt;=100"/>
</Entry>
```

The above-mentioned validation means "lead" parameter of type "double" with default value "52" belongs to group 3 and 4, for which the value must be greater than "0" and less or equal to "100". The "?" means that it is not mandatory to fix the parameter if it is greater than "100". If under 0 the parameter would warn the user and replace the value with 52 while if greater than 100 a warning will be generated but not override would occur.

Example 3

```
<Entry>
  <Key argument="PROMO_AGGR_LEVEL"/>
  <Value type="double"/>
  <Validate group="3,5" restrict=""/>
</Entry>
```

The above-mentioned validation means "PROMO_AGGR_LEVEL" parameter of type double with no default parameter belongs to group 3 and 4, and the validation is done through custom function. The engine will quit running if the validation fails

To add the custom function to the process, you should add your function to *..\Common\Util\Validation Functions.cpp*.

Then add the name and address of this function to the array of function pointers, so that the application can execute this function dynamically:

```
m_mPoint2Function["PROMO_AGGR_LEVEL"]=
PromoAggrLevel;m_mPoint2Function["PROMO_AGGR_LEVEL"]= PromoAggrLevel;
```

Oracle Wallet Troubleshooting

Once the setup process have completed successfully. Verify that the TNS_ADMIN directory contains both tnsnames.ora and sqlnet.ora then follow the steps below to validate the DB connection setup.

Verifying the Wallet Connection on Windows:

1. After completing the setup process successfully CD into the Engine Root directory.

2. Run the setenv.bat batch that was generated by setup process.
3. Run "tnsping %ENG_CONNECTION%" – Tnsping should complete successfully.
4. Run "sqlplus /@%ENG_CONNECTION%" – Sqlplus should connect successfully and allow you to run queries against the configured schema.

Verifying the Wallet Connection on UNIX, Linux or Solaris:

1. After completing the setup process successfully CD into the Engine Root directory.
2. Run the setenv.sh script that was generated by setup process.
3. Run "tnsping $ENG_CONNECTION" – Tnsping should complete successfully.
4. Run "sqlplus /@$ENG_CONNECTION" – Sqlplus should connect successfully and allow you to run queries against the configured schema.

Possible Issues:

- Either tnsping or sqlplus are not found.

  Solution: Make sure you have the ORACLE_HOME environment set up correctly and that you have the ORACLE_HOME\bin directory in your PATH variable.

- Script tnsping fails.

  Solution: Verify that contents of "tns_names.ora" are correct as quoted below.

- Script sqlplus fails to connect

  Verify that contents of sqlnet.ora are correct as quoted below. If they are and the reported error is about incorrect username/password, please redo the setup or recreate the wallet with the correct user/password credentials.

Example TNS_NAMES.ORA File

```
DEM_CONN = (DESCRIPTION= (ADDRESS =
 (PROTOCOL = tcp)
 (HOST = myserver.mydomain.com)
 (PORT = 1521))
 (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = myservice)))
```

Example SQLNET.ORA File

```
SQLNET.AUTHENTICATION_SERVICES = (NTS)
NAMES.DIRECTORY_PATH= (TNSNAMES,EZCONNECT)
WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
%TNS_ADMIN%) ) )
SQLNET.WALLET_OVERRIDE = TRUE
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_VERSION = 0
```

> **Note:** The directory for wallet is pointing to the same directory as set for the environment variable TNS_ADMIN and contains the wallet files (cwallet.soo and ewallet.p12).

Viewing the Engine Log

The log viewer helps you debug the engine run. The log for the Analytical Engine appears in a text file in the directory Demantra_root/Demand Planner/Analytical Engines/bin.

To open the log file viewer:

1. Start the Engine Administrator.
2. Click the View log file button.

To view a log file as it is:

1. Click the Open with Tree View button.
2. When the Processors check box is chosen in Log Groups, you can view the log file with processors tree assistance. If you click on a processor in the right side of the Log File window, you are brought to the corresponding line in the log file.

Examining Engine Results

This section contains assorted tips on viewing and understanding the engine results from a more technical point of view.

Seeing What Level the Forecasting Was Done

When forecasting, the Analytical Engine writes information to the mdp_matrix table to indicate where it performs the forecast. For each combination, it writes this information to the following columns:

- level_id is the strategy in the forecast tree where the forecast for this combination

was generated. Strategy includes data aggregation level referred to as detail node and possible pooling of detail nodes into longer time series referred to as range.

- item_node is the item member in that detail level.
- loc_node is the location member in that detail level.

Seeing if Any Nodes Were Not Forecasted

To see if any nodes failed to receive a forecast, run the following SQL:

```
SELECT level_id, COUNT(*) FROM MDP_MATRIX WHERE prediction_status=1
GROUP BY level_id
```

Explanation: At the start of the run, the engine iterates through all nodes able to be forecasted and sets their level_id to the fictive level. As it forecasts the nodes, it resets the level_id back to normal. At the end of the run, if you have nodes with a level_id = fictive level, those nodes did not get a forecast.

Possible reasons:

- The forecast tree might not be well formed.
- There might not be any models that can work on at the Top Forecast Level.
- There might be nodes that do not have the correct number of observations for the models.
- Naive forecasting might be off; see "Forecast Failure".

Writing Intermediate Results

In a batch run, the Analytical Engine can write intermediate results to the database, to help you determine the source of a problem. To enable this, set the WriteIntermediateResults parameter to yes (1) and then run the engine. When this flag is enabled, the Analytical Engine writes intermediate results to the INTERM_RESULTS table.

> **Warning:** Use this feature only with help of Oracle consulting. This feature may greatly inflate the engine run time.

You can also configure the engine to write forecast data for each node, before splitting to lower levels. This data is written to the NODE_FORECAST table, which includes information on how each model was used for that node. To enable this, set the node_forecast_details parameter to forecast is written with model details (1) before running the engine.

To edit these parameters, use the Business Modeler.

Running the Engine in Recovery Mode

Internally, the Analytical Engine records information to indicate its current processing stage. As a result, if the previous engine run did not complete, you can run recovery, and the Analytical Engine will continue from where it was interrupted.

To run the engine in recovery mode:

1. In the Business Modeler, set the start_new_run parameter to either No or Prompt.
2. Start the Analytical Engine as described in "Running the Engine from the Start Menu".

Stopping the Engine

Normally the Analytical Engine stops on its own when it has completed processing.

If you are automating processes, you may want to make sure that the Analytical Engine is not running, before starting it again.

In the directory Demantra_root/Demand Planner/Analytical Engines, there is a batch file that you can use to kill the engine manager (and therefore the engine as well). This is called KillEngine.bat.

> **Tip:** After killing the Analytical Engine, it is advisable to wait about 10 seconds before starting a new one.

9

Engine Details

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter provides details on the Analytical Engine, for the benefit of advanced users.

This chapter covers the following topics:

- Preparing the Database
- Promotion Effectiveness Engine Phases
- The Forecasting Process
- Comparing Forecast Modes
- Engine Components and High-Level Flow
- Details of the Distributed Engine

Preparing the Database

At the start of an engine run, the Analytical Engine prepares the database, to make sure that the appropriate tables contain rows into which the Analytical Engine can write results. To do so, the Analytical Engine calls the INSERT_UNITS procedure, which is controlled by the RunInsertUnits parameter and can do several things, depending on the value of that parameter:

- Makes sure the engine has rows to write into when generating the forecast. In particular, for *all non-dead* combinations, this procedure does the following:

  1. Checks to see if the database contains records for this combination for all dates in the span of time from max_sales_date to max_sales_date + lead.

  2. For any dates when the combination does not have records, this procedure

inserts records with zero sales, into which the Analytical Engine can then write the forecast.

3. Records with dates in the past are ignored.

- Runs the EXECUTE_PROFILES procedure, which executes the active rolling data profiles.

Additional Details for PE Mode

For Promotion Effectiveness, if the DeleteIsSelfRows parameter is 1, the Analytical Engine also performs a cleaning step. In this step, it removes unneeded rows from the promotion_data, which otherwise can grow to an unreasonable size. (If this table contained a row for every item, every location, every promotion, and every date, performance would suffer.) Specifically, the Analytical Engine deletes rows that have is_self is 0 and that have zero lift values (details below).

In some cases, users may enter override values, and the Analytical Engine should not delete rows that contain those values. The DeleteIsSelfCondition parameter specifies other fields in promotion_data that should be checked before this cleaning occurs. The Analytical Engine deletes only the rows that have is_self is 0 and zero values for all of the following fields: uplift, pre and post-effect, switching effects, and the field or fields specified by DeleteIsSelfCondition.

Promotion Effectiveness Engine Phases

In PE mode, the Analytical Engine runs in multiple phases (the last of which actually generates the forecast), and it caches data at critical points, for better performance. The earlier phases map the promotion attributes internally into causal factors, so that they can be used in the same way as the other causal factors.

This section describes these engine phases.

Global Preparations

This phase uses the following settings from the Promotional Causal Factor screen; see "Configuring Promotional Causal Factors":

Column Name Expression	An expression that retrieves and aggregates the promotion attribute.

Filter	An aggregating expression that returns the true or false value, filtering the source data of this promotional causal factor. You can use this expression to create multiple causal factors from a single set of source data.

When the Analytical Engine runs, the first step is to perform the following global preparations:

- Create the promotion_data_engine table, which is analogous to the sales_data_engine used in demand planning.
- In memory, aggregate the promotion attribute data to the lowest promotional level, as defined in the forecast tree. Here the Analytical Engine uses the Column Name Expression option.
- Apply filters as defined by the Filter option.

Initial Phase

This phase uses the following settings from the Promotional Causal Factor screen; see "Configuring Promotional Causal Factors":

Transpose by Column	Optionally converts a qualitative promotion attribute into multiple unrelated causal factors.
Merge Function	Specifies how Demantra should internally merge promotions of the same kind that apply to the same item, location, and time.
Aggregation Function	Specifies how Demantra should internally aggregate the promotional causal factor above the LPL.

After making global preparations, the Analytical Engine performs the first scan of the forecast tree, as follows:

1. Read from the database and load the forecast tree into memory.
2. Calculate the absolute and relative addressing within each influence group, for internal use. In this step, the Analytical Engine uses the COMPETITION_ITEM and COMPETITION_LOCATION parameter settings.
3. Creating promotional causal factors at the LPL. In this step, the engine does the following:
    - Transpose the promotion attributes, according to the Transpose by Column

option.

- Merge the attributes across promotions, according to the Merge Function option.
- Cache the data for nodes of this level.

4. Creating promotional causal factors at the IGL. In this step, the engine does the following:
   - Aggregate the promotional causal factors within each IG, according to the Aggregation Function field. (If a given promotional causal factor is represented by shapes, those shapes are summed instead.)
   - Cache the data for the IGs.
5. Cache the data for the IRs.

Learning Phase

After the first scan of the forecast tree, the Analytical Engine performs the learning phase, which consists of the following steps:

1. Iterate through the forecast tree, starting at the minimum forecast level.
2. Create the following three historical promotional causal factors for each node in the forecast tree:

self	Influence on this node caused by promotions on this node
own	Influence on this node caused by other nodes within the same IG
other	Influence on this node caused by all IGs within the IR

3. Perform processing to clean up historical data, as specified by various parameters:
   - CutTailZeros
   - ShiftPromoCausals
   - PromotionStartDate
4. Combine the promotional causal factors with the baseline causal factors.
5. Estimate the fit for baseline and promotion coefficients (self, own, and other). If

necessary, discard groups of causal factors for specific combinations.

6. Separately validate the fits for baseline and uplifts.

7. Perform the baseline forecast. This forecast represents the sales without any promotions.

8. Validate the baseline forecast.

9. For any node where the promotion coefficients were validated, partition the uplifts to the promotion attributes that caused them, taking into account the attribute values.

10. Split the baseline and promotional uplifts to the LPL. For lifts, the splitting mechanism does not use the proport mechanism; instead it considers the attribute values, as appropriate. For baseline, proport is used as usual.

11. Decompose the promotional uplifts. In this step, the Analytical Engine associates the uplifts with the specific promotions, rather than the attributes.

12. Compact the promotional uplifts for each combination (combining effects of different promotions). The direct and cannibalization effects are treated separately.

13. For past data, split the fit uplifts to the lowest forecast level (using the normal proport mechanism) and write them to the database.

14. For past data, split the baseline fit and forecast to the lowest forecast level and write them to the database. This step also uses the normal proport mechanism.

15. Cache the forecast level node data.

16. Cache the IDs of relevant forecast nodes to the database.

Promotion Forecast Phase

After the learning phase, the Analytical Engine performs the promotion forecast phase, which consists of the following steps:

1. Iterate the forecast tree, this time only on relevant nodes.

2. Load the forecast node data from the cache.

3. From the cached data, create the future promotional causal factors (self, own, and other) for each node in the forecast tree.

4. Complete the coefficients for future promotional causal factors.

5. Combine the promotional causal factors with the baseline causal factors.
6. Generate the promotional forecast. See "The Forecasting Process".
7. Validate the uplifts. (The baseline has already been validated.)
8. Partition the uplifts, as in the learning phase.
9. Split the baseline and promotional uplifts to the LPL, as in the learning phase.
10. Decompose the promotional uplifts.
11. Compact the promotional uplifts.
12. Split the forecast uplift series to the lowest forecast level and write them to the database.

The Forecasting Process

This section describes the overall forecasting process.

> **Note:** For PE mode, this section describes the process that is performed within the final phase of the engine run; see "Promotion Forecast Phase"

The topics here are as follows:

- Summary of the Forecasting Process
- Preprocessing
- Estimation
- Fit and Residuals
- Validation of Fit
- Causal Factor Testing (Envelope function)
- Forecast
- Engine Split for Future Forecasting
- Validation of Forecast
- Bayesian Blending
- Adjustment

- Forecast Failure
- Intermittent Flow

Summary of the Forecasting Process

The preprocessing module performs the following functions:

1. Cutting leading zeros.
2. (PE mode only) Checking to see whether this node is a promotional node, that is, a combination that has promotions.
3. Deciding whether the node should be treated by the intermittent flow module.
   - (PE mode) First, the node is classified as either promotional or non-promotional, based on whether it has any associated promotions. If the node is promotional, no checking is done for intermittency. If the node is non-promotional, the node is then checked for sparse data; if the node has sparse data, it is flagged for use by the intermittent flow module.

     Note: In later processing, promotional nodes are treated differently from non-promotional nodes in two other ways:

     - The ARIX and ARX models are never used on promotional nodes.
     - The HOLT model is used on promotional nodes only if no other models can be used.

   - (DP mode) If the node has sparse data, it is flagged for use by the intermittent flow module.
4. Treating missing values.
5. Performing preliminary outlier and regime change detection.
6. Removing obvious (gross) outliers, if requested. (This feature is not recommended for use with the engine in PE mode.)
7. Transforming data for use in specific models.

After preprocessing, if appropriate (see Step 3, above), the node is now treated by the Intermittent flow module, which uses special model types; see "Intermittent Flow".

Otherwise, the Analytical Engine applies and tests models as follows:

1. Checking that the number of data points exceeds the number of causal factors by at least two. This is done to ensure that no overfitting will occur, and so that coefficients for all causal factors can be determined.

   The check is valid only for models IREGR, LOG, BWINT, and DMULT. If a model fails this check, it is rejected and a message is written to the log.

2. Estimation. Statistical algorithms are implemented to data and their parameters are calculated.

3. Fit and residuals calculation. The fit reflects the ability of the model to reproduce the actual historical data. The residuals describe the deviation of the fit from the actual data. The results are used later, in the *Bayesian blending method*.

   Then residual outliers are removed, if this option is requested.

4. To check the ability of a model to mimic the actual series, a fit validation is performed (if enabled by the EnableFitValidation parameter). In fit validation, the residuals undergo multiple statistical tests.

5. *Forecast* performs identical calculation to Fit, only for the future period, lead.

6. For a given model, if the forecasting is occurring at the highest forecast level, the Analytical Engine applies a more liberal treatment of models. During forecast validation, models undergo three tests:

   - A test for an unusual zigzag-type jump.

   - A test for abnormal divergence of forecast relative to fit (this is done by building a funnel-shaped envelope and ensuring that the forecast is confined entirely within it).

   - A statistical comparison of forecast and fit means.

   Forecast validation is performed only it is enabled (via the EnableForecastValidation parameter).

7. If at this stage there are no valid models, the time series will be treated by the forecast_failure procedure, where either the control will be passed over to the shell and data accumulated to the next level on the forecast tree, or, if we are already at the top forecast level, the HOLT model will be attempted, if it has not been tried previously as a regular model (and obviously failed). If it has, or if it fails this time, the NAIVE model is fitted (if enabled by the NaiveEnable parameter).

8. On the other hand, if there are valid models, the Analytical Engine applies the *Bayesian blending method*. This combines the results of all the models, taking two factors into account:

   - The variance of the residuals for each model

- The complexity of each model (models that use more causal factors can be overfitted and thus should receive less weighting).

9. It may be necessary to adjust it to pick up the recent trend. The EnableAdjustment parameter directs the flow to the adjustment processor, where trend adjustment is performed, using a set of user-specified parameters.

Preprocessing

The preprocessing stage consists of the following steps:

1. Removing leading zeros. If a series begins with leading zeros, that part of data may be omitted. This is controlled by the CutTailZeros parameter.

2. Intermittency detection and processing. Before checking a series for intermittency, its trailing zeros are temporarily truncated.

   - If there are not enough remaining non zero elements (as measured by the TooFew parameter), the forecast failure module is activated.

   - Otherwise, the IntermitCriterion parameter is checked. This parameter specifies the minimum percentage of zero data points that a series must have to be considered intermittent.

3. Missing values treatment. The Analytical Engine checks the parameter FillParameter. Depending on this parameter null values are replaced by zeros or by the method specified by the FillMethod parameter, which supports the following choices

   - Filling in values by linear interpolation of nearest neighbors.

   - Omitting the values, at the same time adjusting the time scale of causal factors and trends of the Holt procedure. This is useful if you do not want these values not to be accounted for in the estimation procedures. Furthermore, this is the only way to have exact zero "forecasts" in time points where it is known that no demand is expected, like holidays and vacations. Be careful to mark these time points by means of the UpTime parameter.

4. Preliminary outlier detection (if outlier detection is enabled, via the detect_outlier parameter). Outliers are "unusual" data points, that may distort the result of the forecasting process and lead to erroneous decisions. Detecting them is a nontrivial problem. Often what seems to be an outlier turns out to be a result of expected behavior. Even more frequent are cases in which seemingly sound data are in reality outliers.

Note: Outlier detection should be used cautiously with the engine in PE mode. You should not use *gross* outlier detection at all in this mode.

If outlier detection is overused, the engine discards promotions and cannot learn from them. Future promotions will then have no lift.

- The MinLengthForDetect parameter specifies the minimum number of data points needed to perform outlier detection (the default is a year's worth of data).
- Demantra computes a range of "normal" values and counts the number of data points that lie outside that range. If a relatively small number of data points lie outside the range, they are considered outliers and are discarded. On the other hand, if a relatively large number of data points lie outside the range, then Demantra considers all of them to be real data points, and does not discard any of them as outliers. The OutliersPercent parameter controls the threshold for this test.

5. Preliminary outlier handling, of only obvious (gross) outliers. This step is performed only if gross outlier handling is enabled via the GrossRemove parameter . The OutlierStdError parameter controls the sensitivity of the gross outliers detection. The smaller the value, the more aggressively the procedure will be detect outliers.

   Note: At this stage, only the gross outliers are removed. Other outliers are retained, because they may later be attributed to assignable causes, which will be revealed only at the model building stage.

   Gross outlier detection is not recommended for use with the engine in PE mode.

6. Gross outliers are permanently filled by linear interpolation.

7. Preliminary regime change detection (if enabled by the detect_cp parameter). In the preliminary stage, this procedure finds points of change in the level or trend. The RegimeThreshold parameter controls the sensitivity of detection regime change. The smaller the value, the more aggressively the procedure will detect regime changes.

   Note: There is no outlier or regime change detection for intermittent data.

8. If TrendPreEstimation is yes (1), the Analytical Engine performs trend detection.

   Note: If you have disabled negative regression (via UseNonNegRegr), then it is difficult for the Analytical Engine to detect downward trends. In such cases, you should enable trend detection via TrendPreEstimation.

   Trend detection works as follows. The history is divided into two segments: the long segment, which is followed by the short segment. The short segment is assumed to have a trend. Demantra automatically generates a new trend causal factor for each segment (by fitting to the general shapes of those segments) and passes those new causal factors to the engine, to replace the existing trend causals.

   You can specify the following settings to control the specific behavior:

   - First, the TrendPeriod parameter specifies the boundary between the long segment and the short segment. This parameter specifies this boundary in terms of latest, most recent time buckets.

   - The TrendDampPeriod and TrendDampStep parameters specify how this trend should be dampened (toward the future), which is useful particularly with an upward trend (which, when extrapolated, would give unrealistic values). The TrendDampPeriod parameter specifies a block of time (as a number of time periods) over which the residual dampening is applied. Dampening is not applied for the last historical block, and is applied in an exponential manner on previous historical blocks. The size of the dampening depends on parameter TrendDampStep. The TrendDampStep parameter specifies the dampening factor, which is applied n times to the nth block of time. The result is exponential dampening.

   - The TrendModelForShort parameter specifies which engine model to use in order to generate the trend causal factor in the short segment (either REGR or HOLT).

   - The TrendOutlierRatio and TrendShortRatio parameters specify how to treat points found as outliers during trend pre-estimation. Each of these is a numeric weight to apply to the outliers. The TrendOutlierRatio parameter controls the weighting of outliers in the long segment, and the TrendShortRatio controls the weighting of outliers in the short segment.

9. Data transformations for use in specific models.

Estimation

The Analytical Engine uses different estimation procedures for each engine model. See "Theoretical Engine Models".

If UseWeightedRegression is yes (1), then the Analytical Engine applies a weight to each observation when fitting each model. The OBS_ERROR_STD field (in sales_data) specifies the weights for each observation; the default value is 1.

Fit and Residuals

Fit and residual procedures are also model-specific. They calculate values fitted by the model to historical data and evaluate the residuals. Non-positive fitted values are set to zero (depending on the setting of the AllowNegative parameter).

For the logarithmic models (LOG and ELOG), the operation of antilog, to convert results back to original metric, must consider the form of the expectation of a lognormal variable. To use this corrected conversion, activate the LogCorrection parameter.

The Analytical Engine sorts the residuals by size and removes the largest residuals. The parameter RemoveResidOutlier specifies how many residuals to remove, as a percentage of the total number of residuals.

> **Note:** When the engine runs, each analytical model attempts to understand historical demand and then leverages that understanding into generating a future forecast. As part of that analysis, the engine generates "forecast" values that actually occur in the past; these values are called "fit." They are not a direct indication of how accurate future forecasts will be, but are generated to show how well the engine "understands" history.

Validation of Fit

Although fit validation is model-specific, it is activated globally by the parameter EnableFitValidation.

This procedure consists of the following steps:

1. Outliers. Check the influence of outliers on the residuals. The Quantile parameter specifies a standard normal percentile for detecting outliers at a prescribed significance level. If an outlier affects the residuals, no further validation is needed, and we proceed to the problem correction stage. Otherwise, the Analytical Engine tests the goodness of fit.

2. Valid_fit. Here a battery of four statistical tests are performed. Failure of one of them leads to rejection of fit validity.

   - Mean_check is a test for comparison of means of fitted and actual data. The MeanRelativeDistance parameter is the maximum MAPE (Mean Absolute Percentage Error) allowed in a model that is MeanValid.

   - Std_check is a test for comparison of standard deviations of two parts of the

residuals. The division into parts (earlier and later) is controlled by the TestPeriod parameter. The StdRatio parameter is the maximum allowed ratio of the standard deviation of the later part to the standard deviation of the earlier part.

- Bjtest is the Bera-Jarque test for normality of residuals. Normal distribution of errors is a desired feature, assuring randomness, independency and lack of bias in the errors, thus indicating that the model was successful in catching and removing all systematic variability in data.

- Finally, residuals are checked for presence of large deviations, by comparing them to a multiple of standard deviation, as specified by the DeviationFactor parameter.

3. If fit validation fails, the following occurs:

    1. Detect outliers.

    2. Replace the outlying values by values calculated by linear interpolation.

    3. Refit. Re-estimation of model parameters for the series corrected for outliers, recalculation of fit and residuals, followed by revalidation.

Causal Factor Testing (Envelope Function)

For some of the engine models (CMREGR, ELOG, LOG, MRIDGE, and REGR), Demantra can choose random sets of causal factors, which it then tests. Demantra can then either use the set of causal factors that gives the best result or use a mix of causal factors.

This operation is known as the *envelope* function, because it is performed as an envelope around the main engine flow. This operation is controlled by the UseEnvelope parameter, which can equal any of the following:

- 0 (Do not use the envelope function.

- 1 (Use the envelope function on five groups of causal factors: base plus direct and the four switching groups).

- 2 (Use the envelope function on the causal factor groups defined in Estimation_groups table).

- 3 (cycles individual influence groups in and out as part of causal factor envelope analysis process).

Note: A value of '3' is only relevant if at least one active

promotional causal factor is set to 'Has only indirect effects' or 'Has both direct and indirect effects'. For more information, see Influence Group Handling and Filtering.

Additional parameters further control the behavior for specific engine models:

- IGLIndirectLimit specifies the number of influence groups to use when generating cannibalization and halo effects. The top influence groups are chosen based on group volume.

  Note: This parameter is only relevant if at least one active promotional causal factor is set to 'Has only indirect effects' or 'Has both direct and indirect effects'.

- ENVELOPE_RESET_SEED specifies whether to reset the randomization seed for the envelope function, which evaluates different sets of causal factors for different engine models.

- ENVELOPE_CHAIN_LENGTH specifies the number of variations of causal factors to try, for each model.

- BestOrMix specifies whether to use the best set of causal factors (1) or to use a mix of the causal factors (0). The default is 0.

Forecast

The forecast is calculated in almost the same way as the fit; see "Fit and Residuals". The key difference is that the Analytical Engine does not analyze causal factors when computing the forecast. Instead, the engine uses its learning, combined with the future values of the causal factors. The lead parameter specifies the length of time (in the future) for which the forecast is generated. If negative values are disallowed, the Analytical Engine sets them to zero.

Validation of Forecast

At this point, the forecast is validated. The purpose of this validation is to avoid abnormalities in the projected result of a model. The validation is identical for all models, except HOLT, which does not use it. The EnableForecastValidation parameter controls the applicability of forecast validation.

Forecast validation includes three tests:

1. Jump test. This test detects up-and-down or down-and-up zigzag-like jumps. The magnitude of upward jumps is controlled by the Quantile parameter. The larger the value of this parameter, the more liberal is the jump test.

2. Envelope test. This test spreads a funnel-like envelope over the forecast. The shape of the envelope is a function of the behavior of the underlying time series. There is no external control over the sensitivity of envelope test.

3. Mean test is a test on means of the forecast and the later part of the time series of length given by the test_samp_len parameter.

   The ForecastMeanRelativeDistance parameter controls the sensitivity of forecast validation. The larger its value, the more liberal is the test.

Bayesian Blending

First, the Analytical Engine checks the setting of the DetectModelOutliers parameter, which specifies whether to detect model outliers for each forecast node. A model outlier is an engine model that does not give good enough results for that node. The ModelValidationBound parameter controls the sensitivity of the test, which proceeds on each node as follows:

1. For each model, a Demantra proprietary algorithm computes an index that indicates the goodness of fit for that model at that node. Small values are considered good.

2. The Analytical Engine sorts these indexes by value and computes the difference in value between each successive pair of indexes.

3. If none of these differences are greater than the value of ModelValidationBound (whose default is 0.2), the Analytical Engine considers *all* the models good enough and does not look for outliers.

4. If any of the differences are greater than ModelValidationBound, then the Analytical Engine fits a line through the indexes and uses it to determine which models to discard. Any models with points that lie too far above the line are discarded.

For each forecast node, the Analytical Engine discards any model outliers and then combines the results for all models using the Bayesian blending method. This combines the results of all the models, taking two factors into account:

- The variance of the residuals for each model.

- The complexity of each model (models that use more causal factors can be overfitted and thus should receive less weighting).

It is often necessary to enhance models that perform better on most recent historical data, as opposed to models that show close fit to the remote history. This is achieved by assigning decaying weights to residuals, so that recent residuals have greater weights than the remote ones. The DampStep parameter specifies the rate of weights decay, and the DampPeriod parameter specifies the number of periods in which the residuals will

receive the same weights. The dampening of weights is done between each successive period, so that the result is exponential decay.

Adjustment

In the adjustment phase, the Analytical Engine performs a final tuning of the forecast, enabling the user to adjust the forecast to the recent trend in the historical data. Not recommended, unless it is known that a change in trend happened recently, which is likely to be missed by the models. The following parameters are used for adjustment:

- EnableAdjustment enables the adjustment.
- TrendPeriod specifies the period for trend estimation; if zero then no adjustment will be made.
- DownTrend (a value from 0 to 1, inclusive) specifies the degree of descending trend adjustment.
- UpTrend (a value from 0 to 1, inclusive) specifies the degree of ascending trend adjustment.
- PercentOfZeros specifies the maximum percent of zero values in the estimation part to enable trend adjustment.

Forecast Failure

If all participating models fail one of the preceding validations, the control is transferred to the engine shell in order to aggregate to the next level on the forecast tree.

If the model HOLT has not been previously applied at the last level and if there are enough data points, then HOLT is attempted. (HOLT is usually configured for short time series, less than one season). One can optimize its parameters by requesting *Optimally*. The model follows the usual path of estimation, fit and residuals calculation, fit validation, forecast calculation and forecast validation.

If HOLT fails, or if it has been used on this level before, or if there are very little data, an attempt is made to obtain a last resort forecast. Here, the parameter NaiveEnable controls the choice of how to proceed; this parameter has one of the following values:

- no (0): Do not enable either NAIVE or Moving Average models. Do not generate a forecast.
- yes (1): Enable use of the NAIVE model.
- 2 or higher: Enable use of the Moving Average model. In this case, the setting of NaiveEnable specifies the number of recent time buckets to use in calculating the moving average.

If you are using the Analytical Engine in PE mode, note that the NAIVE and Moving

Average models do not generate any lift.

Intermittent Flow

First:

- For PE mode, if a given node has an associated promotion, no checking is done for intermittent data. If it does not have a promotion and if it has as intermittent (sparse) data, it is treated by the Intermittent flow module, which uses special model types.

- For DP mode, if a node has intermittent (sparse) data, it is treated by the Intermittent flow module, which uses special model types.

In the intermittent flow module, the Engine Administrator handles series that were found to be intermittent at the preprocessing stage, according to the IntermitCriterion parameter. Basically, it has many common features with the main flow.

In contrast to the case with non-intermittent models, if there are too many causal factors in comparison with the length of time series, a warning message will be issued, but the model will still be estimated.

The fit validation of intermittent models is simplified and brought down to a comparison of means.

No real forecast validation is done for intermittent models.

If there is a decline in demand at the end of the historical period, then the engine will update the fit after the last spike in history accordingly. To control the intensity of the forecast, you use the IntUpdate parameter.

If the final result is asked for in the form of spikes (as specified by the need_spread parameter), the unspread processor is activated.

The Analytical Engine can run with a minimal set of causal factors. There is no prerequisite for causals in both global and local causal groups. If no global or local causal factors are available, then the Constant global causal factor is used. If the constant causal factor is set to 0, the model could fail with the following message:

```
"Constant should be chosen for both groups of causals. This is strongly
recommended for estimation results, unless sales should be zero for
particular time."
```

The Analytical Engine adheres to the following steps for each causal driven model:

1. Before launching the model, the Analytical Engine builds the matrix ModelGroupCausal from local, global, and PE causal factors. The causal factors are stored in the GroupCausal matrix, and the Analytical Engine picks up only those rows that belong to given model.

2. If no causal factors are available, the model fails with the message "No Causals Available".

3. If the number of available causal factors is more than the number of data points for the forecasted combination, the model fails with the message "Does not have a usable number of observations (too few or too many)."

Influence Group Handling and Filtering

Several parameters control how the Analytical Engine handles influence groups during the evaluation of cannibalization and halo effects. This helps reduce amount of noise encountered by the engine in large influence ranges, and helps limit indirect effects to substantial demand volumes. For example, you can limit the amount of cannibalization generated from the top ten brands.

The IGLIndirectLimit parameter limits the number of influence groups used when generating cannibalization and halo effects.

> **Note:** The IGLIndirectLimit parameter is only relevant if at least one active promotional causal factor is set to 'Has only indirect effects' or 'Has both direct and indirect effects'. UseEnvelope and IGLIndirectLimit parameters will often be used together during a PTP engine run when indirect effects are enabled.

To enable influence group handling:

1. From the Business Modeler, select the Parameters menu and then click System Parameters.
2. Click the Engine tab.
3. On the General subtab, set the UseEnvelope parameter to 3.
4. Click the Save button.

To configure influence group filtering:

1. Depending on your implementation's base time unit, open the appropriate InitParms XML file in a text editor.
   - For daily time definitions, modify **InitParams0Daily.xml**.
   - For weekly time definitions, modify **InitParams0Weekly.xml**.
   - For monthly time definitions, modify **InitParams0Monthly.xml**.
2. Update the value of the IGLIndirectLimit parameter so that the Argument value is the maximum number of indirect influence groups to be analyzed for each node. The value can be '0' (zero) or any positive integer. A value of zero disables the filter. A positive integer allows the highest volume influence groups to participate in halo

and cannibalization. Oracle recommends a value between 5 and 10.

3. Save the file.

Comparing Forecast Modes

For reference, this section compares how the Analytical Engine runs in batch mode, in simulation mode, and in subset forecasting mode.

Batch Mode Characteristics

In a batch run, the Analytical Engine does the following:

1. Traverses a large forecast tree, described in a database. Each node in this tree represents a time-based data series that is a subject to forecast.

2. Performs statistical model calculations on a large subset of the data series (tree nodes). The order of the processing the nodes is important, and is derived from the forecast tree, defined by a few business rules and performance limitations. The forecast tree is traversed using a recursive tree scan.

3. Writes the processed data series to the forecast table in the database.

4. Runs a database procedure that scans and updates the forecast table.

Simulation Mode Characteristics

In a simulation run, the Analytical Engine performs 'what if' scenarios, in which some of the forecast data is changed or different models are run to see how this influences the final results. The four steps related to the batch engine run are also applied here, but on a much smaller section of the forecast tree. The number of data series modeled is much smaller compared to a batch engine run.

Subset Forecasting Mode Characteristics

Subset forecasting incorporates elements of both batch and simulation engines. As in simulation engine, it will process a subset of total data, but as in batch, it will execute without the context of a worksheet and can leverage distributed processing.

In subset mode, engine logs will specify engine is running using subset profile. Engine will note what parent profile the subset profile is associated with as well as display any filter applied to the run.

For example, instead of stating "Running in Batch mode", the log file will state "Running in Subset mode based on Parent Profile XXXX" and "Filter YYYY is applied to engine population."

Important: It is strongly recommended the population which the engine runs on will be filtered using PopulationExtraFilter. For more information, refer to description in "Analytical Engine Parameters".

Important: When running engine in subset mode, it is strongly recommended parameter RunInsertUnits be set as "run nothing", as this should typically be executed during batch runs. If no full batch run is planned, this parameter can be set to active (1 or 3).

Example:

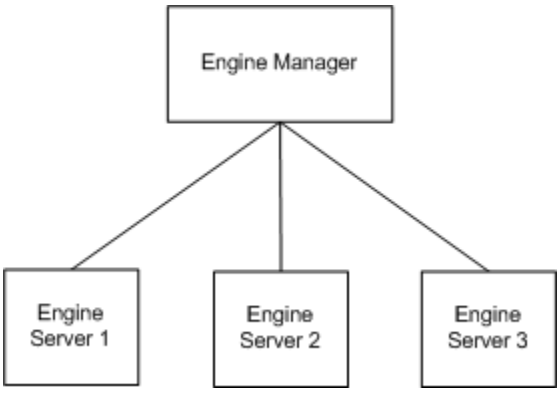
- System has 100 active combinations
- Engine last executed in batch mode using batch engine profile.
- Batch profile has two forecast versions, Fore_4 column holds currently active version and Fore_2 holds an older archive version of forecast.
- When engine is run again using the batch profile, the oldest available column will be used to store newly generated forecast. Any other forecast column will serve as an archive of newly generated forecast. The forecast will be written to Fore_2 column for all 100 combinations and Fore_4 column will be left alone. Fore_2 now holds the active version and Fore_4 holds the older archive.
- A Subset engine profile is created with name NewProds and is associated with parent profile batch.
- Profile NewProds is configured to execute on only 10 combinations.
- When engine is executed using NewProds profile, forecast will be regenerated for the filtered 10 combinations and forecast will be rewritten to Fore_2 column. Forecast for 90 combinations filtered out will not be modified.

Engine Components and High-Level Flow

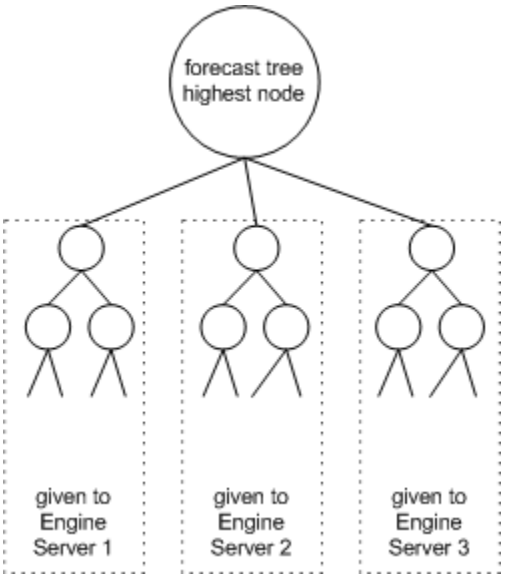
At a higher level, it can be useful to understand how the Analytical Engine divides and processes its work.

Engine Components

Internally, the Analytical Engine consists of one Engine Manager and multiple Engines.



The engine server scans a portion of the forecast tree, and sends the output to the proport mechanism. The engine server masks the mdp_matrix table and processes only the nodes that are in the part of the tree relevant to its task. The ID of the task is received from the Engine Manager, which is responsible for dividing the forecast tree into smaller sub trees (called tasks).



The Engine Manager is responsible for controlling the run as a whole. Communication between the various engine modules is achieved by use of Oracle advanced queue notifications.

Engine Components and Batch Run

The following steps describe the responsibilities of each component during a batch run of the Analytical Engine.

1. The Engine Manager starts Engines via EngineStarter. The startup process includes the following steps:

- Engine Manager creates a notification listener. The Engines will use this interface to make requests for new tasks to process, or to return status completion information to the Engine Manager.

- The Engine Manager passes the database settings and all other settings to the Engines.

- The Engines connect to the database and load parameters.

- The Engines initialize themselves using the xml schema files and request the Engine Manager for tasks to process.

2. The Engine Manager checks if the run is a recovery run or a new run, and acts accordingly. If it is a recovery run, the Engine Manager retrieves unfinished tasks. If it is a new run, the Engine Manager resets the mdp_matrix table and allocates a forecast column. The Engine Manager divides the forecast tree into smaller tasks by updating one column in mdp_matrix that links each node with a task ID. The number of the tasks that the Engine Manager attempts to create is the number of Engines that were initialized successfully, multiplied by a configurable factor.

3. The Engine Manager executes all the Engines and waits for them to return a final completion status.

4. When an engine server is executing, it uses the Engine Manager callback interface in order to get task IDs to process (pull approach). The data flow between the Engine Manager and the Engines is very low volume, containing only settings, task IDs and statuses. The data that flows between the Engines and the database includes the sales (input) and forecasted (output) data (very high volume), forecast tree configuration information, database parameters, and certain other information.

5. The engine server uses the task ID to create a sales_data_engine table (or view) with the records for that task and then scans the forecast tree, operating select and update queries on the mdp_matrix table. During the processing of a task, an engine server filters mdp_matrix according to the task ID and operates only the subtree relating to that task. It uses two threads, one for scanning the tree and performing calculations, and one for the proport mechanism.

6. When the engine server gets a null task ID from the Engine Manager, it knows that no more task IDs are available, and it sends a completion notification to the Engine Manager.

7. When the Engine Manager has received a completion status indicator from all the Engines, it updates the run status, executes the post process procedure, and the engine run is completed.

Details of the Distributed Engine

Your system may include the Distributed Engine, which is a mode in which the Analytical Engine automatically distributes its work across multiple machines simultaneously.

> **Note:** For the Distributed Engine to work, the Analytical Engine must be registered on multiple machines, all of which have database client software in order to access the Demantra database.

The Distributed Engine drastically shortens the run time for a single batch engine run by processing the engine tasks in parallel, on *different machines*, for improved engine processing time. Also, multiple simulation requests can be handled simultaneously.

In a batch run, the Distributed Engine starts by reading a settings file that lists the machines on the network where the Analytical Engine is installed. The Engine Manager tries to instantiate an engine server on the machines in this list. Processing then continues with Step 1.

10

Engine Parameters

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter describes the Analytical Engine parameters that you can see in Business Modeler and lists their default values, if any.

This chapter covers the following topics:

- About Engine Parameters
- Analytical Engine Parameters

About Engine Parameters

For each parameter, this chapter indicates which engine variations that parameter can be used with. This chapter also indicates which parameters can be used with nodal tuning. Some of the Promotion Effectiveness (PE) parameters are useful only if your system also includes Promotion Optimization.

Oracle provides two different modes for the Analytical Engine:

- In PE mode, the engine is suitable for use with Promotion Effectiveness.
- In DP mode, the engine is suitable for use in demand planning applications.

As indicated, most parameters are visible to all users; a few are visible only if you log in as the owner of the component.

See also

"Theoretical Engine Models"

Analytical Engine Parameters

Parameter	Location	Default	Engine Mode*	Details	Tuning
A					
add_zero_combos_to_mdp	Engine > Data Manipulation	yes	Both**	**Visible only to owner.** Specifies the Proport mechanism handles combinations whose historical data consists of zeros. Use one of the following values: • yes: Add these combinations to mdp_matrix even if their historical data consists of zeros. • no: Do not add these combinations.	
AllowableExceptions	Engine > Validation	10	PE only	**Visible only to owner.** Specifies the permissible amount of exceptional uplifts, as a percentage of total number of uplifts. The LowerUpliftBound parameter controls the threshold for exceptional uplifts. The engine discards a model (for a given forecast node) in either of two cases: • If the model generates too many exceptional uplifts (as specified by the LowerUpliftBound and AllowableExceptions parameters). • If any uplift exceeds the bound given by the UpperUpliftBound parameter.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
AllowNegative	Engine > Adjustment	no	Both	This parameter is used by the fit and residuals module of the Analytical Engine. Use one of the following values: • yes: Negative values of fit and forecast are allowed. • no: Any non-positive fitted and forecasted values are set to zero.	Can be tuned by node
AnalyzeMdp	Engine > Shell	Full analyze	Both	**Visible only to owner.** Specifies how to analyze the mdp_matrix table after the Engine Manager divides the forecast tree into tasks. Use one of the following values: • 5 columns analyze: Enable a partial analysis using the five most important fields: prediction_status, prop_changes, branch_id, do_aggri, and do_fore. • Full analyze: Enable a full analysis. • No analyze: Disable the analysis. **Note:** The branch_id field is for internal use only.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
AverageHorizon	Engine > Data Manipulation	12 for monthly 52 for weekly 7 for daily	PE only	**Applies only to Promotion Optimization; parameter is visible only to owner.** Specifies the length of time to be used in calculating the average baseline forecast. This window of time starts at the date given by the StartAverage parameter. For information on configuring Promotion Optimization, see "Configuring Promotion Optimization for PTP" in the *Oracle Demantra Implementation Guide*.	Global setting only
B					
BatchRunMode	Engine > Shell	estimation and forecast run	PE only	Specifies the kind of forecasting to do: • run the forecast against only the learning (0; estimation) • run the promotion forecast (1; recommended setting) • estimation and promotion forecast run (2; fast simulation), using previously cached data. If no cached data is found, the Analytical Engine gives a message and calculates the needed data. This parameter applies to both batch run and simulation run.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
BottomCoefficientLevel	Engine > Data Manipulation	1	PE only	**Applies only to Promotion Optimization; parameter is visible only to owner.** Specifies the lowest forecast tree level for which the Analytical Engine will calculate coefficients. Use any forecast tree level between the lowest promotional level and the InfluenceRangeLevel, inclusive. For information on configuring Promotion Optimization, see "Configuring Promotion Optimization for PTP" in the *Oracle Demantra Implementation Guide.*	Global setting only
BulkLoaderBlockSize	Engine > Shell		Both	**Oracle only; visible only to owner.** Specifies the minimum amount of number of rows that the Analytical Engine loads at one time, when writing to the database. The larger this is, the more quickly the data is loaded, but there is greater risk if the database connection is lost. Use a value between 100 and 100,000.	Global setting only
BulkLoaderEnableRecovery C	Engine > Shell		Both	Specifies whether Oracle Bulk Loader should perform recovery after a lost database connection. Oracle Bulk Loader is used by the Analytical Engine.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
CachePath		Null	Both	Specifies the path to the directory into which the Analytical Engine should write its caching files. This can be any of the following: • A relative path (relative to Demantra_root/Demand Planner/Analytical Engines/bin). • An absolute path. • Null. In this case, the Analytical Engine creates its caches in Demantra_root/Demand Planner/Analytical Engines/bin/cache. You should create the directory manually if it does not yet exist.	Global setting only
CalcOptimizationInput	Engine > Data Manipulation	no	PE only	**Applies only to Promotion Optimization; parameter is visible only to owner.** Specifies whether the Analytical Engine should calculate inputs needed for Promotion Optimization. Use one of the following values: • yes (1): See "Configuring Promotion Optimization for PTP" in the *Oracle Demantra Implementation Guide*.. Make sure to set the IS_OPTIMIZATION flag equal to 1 for at least one of the linear engine models. • no (0)	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
cannibalism	Engine > Data Manipulation		Both**	Specifies the default values for aggri_98 and aggri_99, which are combination-specific fields. If equal to 0 or 1, the defaults for both fields are 1. If equal to 2, the default for aggri_98 is 1, and the default for aggri_99 is 0.	Global setting only
CannibalizationIgnore	Engine > Data Manipulation		PE only	Controls whether the Analytical Engine will calculate switching effects (cannibalization). You can use this parameter to easily switch off that calculation when needed, for example, when running specific simulations.	Global setting only
Collinearity MaxRatio	Engine> Data Manipulation	5	Both	Maximum ratio allowed between base or lift elements and total demand. Can be set to any integer greater than one.	Global setting only
CollinearityTolerance	Engine> Data Manipulation	1	Both	Parameter controlling sensitivity of collinearity detection. Default value of 1 detects very strong cases where large values such as 1000 would detect weaker cases. Can be set between 1 and 100,000.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
CollinearityUseRidge	Engine> Data Manipulation	1	Both	This flag controls the use of Ridge regression or QR/SVD decomposition in cases of collinearity. Possible values are: • 0 - Ridge Service is not used • 1 - Use Ridge for cases of collinearity. • 2 - Use QR instead SVD for cases of collinearity and NonNegRegr.	Global setting only
COMPETITION_ITEM	Engine > Shell		PE only	**Visible only to owner.** Specifies the level (from the group_tables table) that defines the competitive item (CI) groups. Each node of this level represents a different item group. The CI should be consistent with the item groups (I). Specifically, two items within a given item group must also belong to the same competitive item group. The easiest way to follow this rule is to set the CI equal to an item level that is higher than I and that is within the same hierarchy. A similar rule applies for the locations. **Note:** You specify the item groups indirectly when you specify the IGL in the forecast tree. see "Configuring the Forecast Tree".	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
COMPETITION_LOCATION	Engine > Shell		PE only	**Visible only to owner.** Specifies the level (from the group_tables table) that defines the competitive location (CL) groups. See the notes for COMPETITION_ITEM.	Global setting only
CutTailZeros D	Engine > Data Manipulation	yes	Both	**Visible only to owner.** Specifies how the preprocessing module (of the Analytical Engine) should handle series that start with zero values. Use one of the following values: • yes: Delete the leading zeros. • no: Retain them as actual zero values.	Can be tuned by node
DampPeriod	Engine > General	0	Both	This parameter is used by the Bayesian blending module of the Analytical Engine. It specifies the length of periods in which the residuals will receive the same weights. The dampening of weights is done between each successive period. This parameter lets you put greater weight on models that perform better on most recent historical data, as opposed to models that show close fit to the remote history.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
DampStep	Engine > General	0	Both	This parameter is used by the Bayesian blending module of the Analytical Engine. It specifies the rate of weights decay. By setting this parameter to 0 (or 1), you set all weights to be equal to 1 (equal weights).	Can be tuned by node
def_delta	Engine > Proport	0.75	Both**	Specifies the default value for the delta field in the mdp_matrix table. If delta equals null for a given combination, the system uses the value of this parameter instead. All new combinations created through data loading, member management, and/or chaining will have a null value in their delta column, thus indicating that they will also take the default delta value from this parameter. In turn, the delta field is used in the proport calculation as in the following example: P1 = glob_prop * delta + (monthly demand) * (1 - delta)	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
DeleteIsSelfRows			PE only	Specifies whether the Analytical Engine deletes unneeded promotion_data records. Use one of the following values: • 0 means that the Analytical Engine does not delete records in promotion_data. • 1 means that the Analytical Engine deletes unneeded records. A record is considered unneeded if all the following conditions are true: It is flagged as is_self = 0 All lifts (uplift, pre and post effect, and switching effects) equal 0 The condition specified by DeleteIsSelfCondition is true Also see "Is_Self".	Global setting only
DeleteIsSelfCondition			PE only	Specifies an additional true/false condition that must be met to delete unneeded records in promotion_data. Used only if DeleteIsSelfRows is 1. This parameter is used as an SQL extra where clause. The Analytical Engine uses it to restrict the deletion.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
detect_cp	Engine > Outlier and Regchange	yes	Both	This parameter is used by the preprocessing module of the Analytical Engine. Use one of the following values: • yes: The engine should attempt to detect a regime change in the level or trend. If it finds a change point, it performs the analysis on the leveled out series. The threshold for change points is controlled by the RegimeThreshold parameter. • no: The engine should not attempt to detect change points. The RegimeThreshold parameter is ignored.	Can be tuned by node
detect_outlier	Engine > Outlier and Regchange	yes	Both	This parameter is used by the preprocessing module of the Analytical Engine. Use one of the following values: • yes: The engine should attempt to detect outliers. If it finds outliers, it considers them in the analysis. • no: The engine should not attempt to detect outliers. Also see GrossRemove. To disable all outlier detection, both these parameters must be switched off.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
DetectModel Outliers			Both	**Visible only to owner.** Specifies whether to check for outlier models for each forecast node. Outlier models are models that do not fit well enough. The sensitivity of the test is controlled by the ModelValidationBound parameter.	Global setting only
DeviationFactor	Engine > Validation	5	Both	**Visible only to owner.** This parameter is used by the fit validation module of the Analytical Engine, and it controls the sensitivity of one of the fit validation tests. In this test, residuals are checked for presence of large deviations, as specified by DeviationFactor. This parameter specifies the maximum number of standard deviations that the residuals are allowed to attain. A model is rejected if it fails this test.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
DownTrend	Engine > Adjustment	0.2	Both	This parameter is used by the adjustment module of the Analytical Engine, if that module is enabled (via EnableAdjustment). It controls the forecast adjustment for downward trend. Specifically, it specifies the amount by which the forecast is rotated to align with recent trend in data. Use a value from 0 to 1, inclusive. Enabling adjustment is not recommended, unless it is known that a change in trend happened recently, which is likely to be missed by the models.	Can be tuned by node
dying_time	Engine > Proport	0.5 season (1 season in media)	Both**	If no sales occurred during the length of time specified by this parameter, the combination is marked as dead. See prediction_status. Global setting, but may also be defined locally in a worksheet.	Global setting only
E					

Parameter	Location	Default	Engine Mode*	Details	Tuning
EnableAdjustment	Engine > Adjustment	no	Both	This parameter controls the adjustment module of the Analytical Engine. Use one of the following values: • yes: Enable the adjustment module, which performs a final tuning of the forecast, adjusting the forecast to the recent trend in the historical data. • no: This is the recommended setting, unless you are sure that a change in trend happened recently, which is likely to be missed by the models.	Can be tuned by node
EnableFitValidation	Engine > Validation	yes	Both	**Visible only to owner.** This parameter controls the fit validation module of the Analytical Engine. Use one of the following values: • yes: Perform a normal validation for the fit. • no: Perform only a weak validation.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
EnableForecastValidation	Engine > Validation	yes	Both	**Visible only to owner.** This parameter is used by the forecast validation module of the Analytical Engine. Use one of the following values: • yes: Perform a normal validation for the forecast. • no: Perform only a weak validation.	Can be tuned by node
EnableModifiedVariance	Engine > General	no	Both	**Visible only to owner.** This parameter is used by the fit validation module of the Analytical Engine. Use one of the following values: • yes: Perform the modified variance, which specifies how the variance is calculated in determining weights for Bayesian blending. • no	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
EnableSimGL Filter	Engine > General	yes	PE only	**Visible only to owner.** Specifies whether simulation should respect or ignore any general-level filtering applied to the worksheet. Use one of the following values: • yes: Respect the general level filter and run the simulation only on combinations in the worksheet and only on the general level members that is included in the filter. This option ignores, for example, any other general level members associated with those combinations. This setting should be used for fast simulations only. If used on promotions or scenarios, only the selected member will receive a regeneration of uplift. All other members—even if they would normally interact with each other—will be excluded. If learning is run using this setting, there is a very good chance that engine results will be wrong due to inclusion of only a part of history. • no: Ignore the general level filter and potentially run the simulation on combinations that are not included in the worksheet. This is the previous behavior.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
				This parameter has no effect if the worksheet is not filtered by a general level.	
EngDimDef_ItemOrLoc	Engine > Tables	none	DM only	Determines which forecast tree dimension, item or location displays additional levels. The default is that no additional levels are shown.	Global Setting only
EngineOutputThreshold	Engine > General	0	Both	see Fine Tuning and Scaling Demantra -- EngineOutputThreshold	Global Setting only
EngineScaleInput	Engine>General	0	Both	Activates engine scaling of demand and causals as necessary when viewing weekly data by calendar month. If set to No (0-default) no scaling will be done. If set to Yes (1) demand periods and causals are scaled by the forecast engine.	Global Setting Only
EngineScaleIntermInput	Engine>General	0	Both	Controls whether to scale data Intermittent nodes when scaling data. This parameter only applies when EngineScaleInput set to Yes. If set to No (0-default), no scaling will be done. If set to Yes (1), scaling will occur for intermittent forecast nodes and the Croston model is disabled.	

Parameter	Location	Default	Engine Mode*	Details	Tuning
EngKeyDef_Supersession	Engine > Proport	item_id, location_id	DM only	Key used to aggregate members belonging to the same supersession set. When set to the same value as EngKeyDefPK, the proport calculates proportions for each lowest-level member processed by the engine individually and no special handling of supersessions is done. If set above the level defined by EngKeyDefPK, then calculation of proportions is done at this aggregated level considered the supersession and all underlying combinations receive the same proportional values.	Global Setting only
EngKeyDefPK	Engine > Tables	item_id, location_id	DM only	Defines the primary key of the combination and data tables selected for this Analytical Engine profile.	Global Setting only
EngTabDef_HistoryForecast	Engine > Tables	SALES_DATA	DM only	Table that holds historical demand and into which the forecast to be written.	Global Setting only
EngTabDef_Inputs	Engine > Tables	INPUTS	DM only	Table that contains time definitions for the chosen engine profile. This parameter should not be modified.	Global Setting only
EngTabDef_Matrix	Engine > Tables	MDP_MATRIX	DM only	Table that holds the combinations available for the chosen engine profile.	Global Setting only
EngTabDef_Parameters	Engine > Tables	PARAMETERS	DM only	Table that holds the analytical model parameters for engine profiles.	Global Setting only
F					

Parameter	Location	Default	Engine Mode*	Details	Tuning
FillMethod	Engine > Data Manipulation	linear interpolation	Both	This parameter is used by the preprocessing module of the Analytical Engine (if FillParameter equals 1). The FillMethod parameter specifies how to fill any null (missing) values. Use one of the following values: • linear interpolation: Fill in values by linear interpolation of nearest neighbors. • omitting missing values: Omit the null values and adjust the time scale of causal factors and trends of the Holt procedure; also see the UpTime parameter. • This parameter is ignored if FillParameter equals 0.	Can be tuned by node
FillParameter	Engine > Data Manipulation	0	Both	This parameter is used by the preprocessing module of the Analytical Engine. It specifies how to handle null (missing) values. Use one of the following values: • yes: • no: • If equal to 0, null values are replaced by zeros and FillMethod is ignored. • If equal to 1, null values are filled as specified by FillMethod.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
ForecastGenerationHorizon	Engine > Time	0	Both	Specifies what historical fit data the engine will write to the database. If this parameter is 0, the engine writes the forecast only. If this parameter is a positive integer N, the engine writes the last N historical fit values.	Global setting only
ForecastMeanRelativeDistance G	Engine > Validation	3.5	Both	**Visible only to owner.** This parameter is used by the forecast module of the Analytical Engine. It specifies the sensitivity of forecast validation. The smaller the value, the stricter the test.	Can be tuned by node
GLPropSuperSessionMethod	Engine > Proport	latest revision	DM only	Defines the method general level proportions use to allocate proportions during supersessions. When set to the default for each period, proportions are allocated completely to the member with the latest starting date. If set to All Active Revisions for each period, proportions are allocated equally among all active members.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
GrossRemove	Engine > Outlier and Regchange	No	Both	This parameter is used by the preprocessing module of the Analytical Engine. Use one of the following values: • yes: The engine should process gross outliers. Enable this feature only if there is a clear reason to remove obviously unreasonable values. The threshold for gross outliers is controlled by the OutlierStdError parameter. • no	Can be tuned by node
H					
HighestSquaring	Engine > Validation	4	Both	**Visible only to owner.** This parameter is used by the fit validation module of the Analytical Engine. It specifies the number of residual standard deviations, beyond which the residuals participate in the sum of squares calculation in their absolute value, rather than squared.	Can be tuned by node
hist_glob_prop	Engine > Proport	1 season	Both**	Maximum number of base time buckets to use in calculating glob_prop, the running average demand for any given item-location combination. This parameter is used by the proport mechanism. Global setting, but may also be defined locally in a worksheet.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
HistoryLength	Engine > Time	0	Both	The number of base time buckets to consider for fit estimation and for the proport mechanism. Must be a non-negative integer. If equal to 0, the length of the history is set by the start_date parameter instead.	Can be tuned by node
I					
InfluenceGroupLevel	Engine > Shell		PE only	**Read-only.** Specifies which level (from the group_tables table) is used as the influence group level of the forecast tree. To specify this parameter, you use the Forecast Tree Editor within the Business Modeler.	Global setting only
InfluenceRangeLevel	Engine > Shell		PE only	**Read-only.** Specifies which level (from the group_tables table) is used as the influence range level of the forecast tree. To specify this parameter, you use the Forecast Tree Editor within the Business Modeler.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
IntermitCriterion	Engine > General	99	Both	This parameter is used by the preprocessing and intermittent flow modules of the Analytical Engine. It specifies the minimum percentage of zero data points that a series must have to be considered intermittent. In this test, leading zeros may or may not be considered (depending on the setting of CutTailZeros). Trailing zeros are ignored in either case. In the extreme case where this parameter equals 0, all series are treated as intermittent.	Can be tuned by node
IntUpdate	Engine > Adjustment	0.5	Both	This parameter is used by the intermittent flow module of the Analytical Engine. It specifies the degree to which the Analytical Engine will update the fit after the last spike in history, in the case where there is decline in demand at the end of historical period. Use a number between 0 and 1, inclusive. The value 1 means that the change in fit is to be carried forward fully to the forecast. On the other extreme, the value 0 means that no change is to be applied. A value between 0 and 1 will be used as a weight for combining past and updated behavior.	Can be tuned by node
L					

Parameter	Location	Default	Engine Mode*	Details	Tuning
last_date	Engine > Time	1/1/1900	Both	Last date of actual sales, to be used by the Analytical Engine and the proport mechanism. No dates after this are used towards the forecast or the proport calculation. If this parameter equals 1/1/1900, the system instead uses last_date_backup.	Global setting only
last_date_backup	Engine > Time		Both	Specifies a backup value to use for the last sales date, in case last_date is 1/1/1900. Sometimes, when you load sales data, you need to change this parameter so that you can ignore a recent subset of history. The proport mechanism makes sure that this parameter is never later than max_sales_date. See "max_sales_date".	Global setting only
lead	Engine > Time	12 for monthly data, 52 for yearly, 30 for daily	Both	The number of base time buckets to predict. The Analytical Engine generates a forecast for the base time buckets in the span from max_sales_date to max_sales_date + lead. See "max_sales_date".	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
LogCorrection	Engine > General	no	Both	This parameter is used by the fit and forecast modules of the Analytical Engine. The issue is that logarithmic models use log-transformed demand data, which can give inaccurate results if that transformed data is near to 1 in value. In such a case, you may want to use this parameter to make an internal adjustment. Use one of the following values: yes: Use correct form of the expectation of a lognormal variable. no: Do not perform the log correction.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
LogLevel				Controls the amount of detail that is written into the Analytical Engine log. Use one of the following values: 1. Critical 2. Error 3. Warning 4. Message **Note:** This corresponds to the amount of detail that the log has contained in past releases. 5. Info 6. Detail This setting applies to all log groups chosen through the Engine Administrator:	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
LowerUpliftBound M	Engine > Validation	3	PE only	**Visible only to owner.** Specifies the limit beyond which an uplift value is considered "exceptional." This limit is specified as a proportion of baseline. For each model, the Analytical Engine compares the absolute value of the uplift, divided by baseline, to this parameter. The engine discards a model (for a given forecast node) in either of two cases: • If the model generates too many exceptional uplifts (as specified by the LowerUpliftBound and AllowableExceptionsparameters). • If any uplift exceeds the bound given by the UpperUpliftBound parameter.	Can be tuned by Node
mature_age	Engine > Data Manipulation	2	Both**	Controls the mature_date of each combination, which is calculated backwards from the current date using the mature_age parameter. A combination is young (rather than active) if it does not have any non-zero sales data on or before the mature_date. See prediction_status. This is a global setting, but may also be defined locally in a worksheet.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
max_accept_num				Maximum absolute value that is permitted for the forecast results. If the Analytical Engine generates a result larger than this in absolute value, it substitutes this maximum (or minimum, if applicable). **Tip:** Make sure the forecast columns are large enough to accommodate a number of this size, and be sure to account for a possible negative sign. Errors occur if the Analytical Engine cannot write the forecast because the database columns are not large enough.	Global setting only
max_fore_level	Engine > Shell	Level just below the highest fictive level	Both	The maximum level on the forecast tree at which a forecast may be produced. Upon failure at this level, the NAIVE model will be used (if NaiveEnable is yes). In Promotion Effectiveness, this must be at or below the influence range level (IRL); see InfluenceRangeLevel.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
MaxEngMemory	Engine > Proport	100	Both	Specifies the maximum amount of system memory usage (in megabytes) at the end of a task, before an engine is re-initiated. Use this parameter to prevent the Analytical Engine from failing or generating errors when processing large amounts of data. For machines that run multiple engines, ensure that each engine has access to the amount of memory specified.	Global setting only
MeanRelativeDistance	Engine > Validation	0.5	Both	**Visible only to owner.** This parameter is used by the fit validation module of the Analytical Engine, and it controls the sensitivity of one of the fit validation tests. A model is rejected if its MAPE (Mean Absolute Percentage Value) is greater than this threshold. The smaller the value, the stricter is the validation.	Can be tuned by node
MetricsPeriods	Engine > Validation	26	Both	Number of recent periods used to calculate automated engine accuracy metrics. If set to 0 engine will not generate metrics.	Tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
min_fore_level	Engine > Shell	1	Both	Minimum forecast level that the engine will forecast. From that level down, the engine will split the forecast using the precalculated proportions in the mdp_matrix table. For PE, this must be at or above the lowest promotional level (LPL).	Can be tuned by node
MinLengthForDetect	Engine > Outlier and Regchange	12 for monthly data, 52 for weekly, 14 for daily	Both	This parameter is used by the preprocessing module of the Analytical Engine. It specifies the minimum number of data points needed in order for the engine to try to detect outliers and regime changes.	Can be tuned by node
ModelValidationBound		0.2	Both	Specifies the sensitivity of the test used to detect "outlier" models for a given node. Outlier models are models that do not fit well enough. This parameter is used only if model outlier detection is enabled (via the DetectModelOutliers parameter.)	Global setting only
N					

Parameter	Location	Default	Engine Mode*	Details	Tuning
NaiveEnable	Engine > General	yes	Both	Specifies what to do at the highest forecast level, upon failure of all models. Use one of the following values: • no (0): Do not enable either NAIVE or Moving Average models. Do not generate a forecast. • yes (1): Enable use of the NAIVE model. • 2 or higher: Enable use of the Moving Average model. In this case, the setting of NaiveEnable specifies the number of recent time buckets to use in calculating the moving average.	Can be tuned by node
need_spread	Engine > Adjustment	produce continuous forecast	Both	This parameter is used by the intermittent flow module of the Analytical Engine, and it controls whether the final result should be given in the form of spikes. Use one of the following values: produce forecast with spikes produce continuous forecast This applies only to intermittent models.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
node_forecast_details	Engine > Shell	forecast is written to node_forecast_q	Both	**Visible only to owner.** Specifies whether the Analytical Engine should write forecast data for each node, before splitting to lower levels. Use one of the following values: • forecast is written with model details (1): The Analytical Engine writes intermediate forecast data for each node, to the NODE_FORECAST table. The table includes information on how each model was used for that node. The Analytical Engine will run more slowly because of the additional work in writing to this table. • forecast is written to node_forecast_q (0): The Analytical Engine writes the forecast as usual.	Global setting only
NonNegRegrMaxTolMult				Specifies the maximal multiplier to be used in order to increase the tolerance value in nonnegative regression. When you disable negative coefficients (via UseNonNegRegr) and are unable to acquire a solution, it may be helpful to increase this tolerance. Recommended value range: 30 - 2000 Default value: 30	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
NormalizationFactor	Engine > Data Manipulation	0	PE only	**Parameter is visible only to owner.** Specifies the degree of normalization to perform, if NormalizeResults is yes. Use a number from 0 to 1, inclusive. The ends of this range have the following meanings: • 1 means preserve the baseline fit. In this case, all residuals are added to the uplift. • 0 means that both the baseline and uplift are modified according to the normalization algorithm. This is the recommended setting. This normalization is applied only to historical data (where the baseline is known).	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
NormalizeResults	Engine > Data Manipulation	no	PE only	**Parameter is visible only to owner.** Specifies whether to normalize the historical engine results so that the observed baseline values are preserved. Use one of the following values: • yes (1): Normalize historical engine results so that the observed baseline values are preserved. In this case, the Analytical Engine writes these results into the columns fore_a_normal, etc. This setting is recommended for use when historical analysis is of importance. Will cause Base + Lift to exactly match demand (quantity_form). The results are written in different column from base and lift to enable ease of comparison. No normalized results are available for future dates, because of the lack of normalization number; this potentially makes the connection between historical and future forecast not smooth. • no (0): Do not perform normalization.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
NumShapes O	Engine > Validation	8	Both	**Parameter is visible only to owner.** Specifies the maximum number of allowed shape causal factors for the engine to use for a given node in the forecast tree. Use an integer from 0 to 8, inclusive. Applies to activity shape modeling (rather than to promotional shape modeling).	Global setting only
oracle_optimization_mode	Engine > Shell	cost	Both	**Oracle only; visible only to owner.** Optimization mode of the database.	Global setting only
OutliersPercent	Engine > Outlier and Regchange	25	Both	This parameter is used by the preprocessing module of the Analytical Engine. A set of points, suspicious as outlying, will be regarded as such only if its size does not exceed this given percentage of data.	Can be tuned by node
OutlierStdError P	Engine > Outlier and Regchange	2.5	Both	This parameter is used by the preprocessing module of the Analytical Engine, if gross outlier processing is enabled (via the GrossRemove parameter). The OutlierStdError parameter specifies the sensitivity of gross outlier detection. The greater this value, the less sensitive (more liberal) is detection of gross outliers. The value 0 is not allowed.	Can be tuned by Node

Parameter	Location	Default	Engine Mode*	Details	Tuning
PartitionColumnItem			Both	Specifies the name of the column that partitions the data by item. This column must exist in sales_data, mdp_matrix, and (for Promotion Effectiveness) promotion_data. If this is null, data is not partitioned by item. See "Database Partitioning for the Engine".	Global setting only
PartitionColumnLoc			Both	Specifies the name of the column that partitions the data by location. This column must exist in sales_data, mdp_matrix, and (for Promotion Effectiveness) promotion_data. If this is null, data is not partitioned by location. See "Database Partitioning for the Engine".	Global setting only
PercentOfZeros	Engine > Adjustment	0.2	Both	This parameter is used by the adjustment module of the Analytical Engine, if that module is enabled (via the EnableAdjustment parameter). It specifies the maximum fraction of zero values in data beyond which no forecast adjustment is performed. Use 0.2 for 20 percent, for example. Enabling adjustment is not recommended, unless it is known that a change in trend happened recently, which is likely to be missed by the models.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
PopulationExtraFilter	Engine> Shell	Null	Both	This parameter can be used to apply an extra filter on the combination population processed by the engine. This can help support incremental engine runs as well as ensure only desired combinations are forecasted. The default setting is null, applying no additional filter. An expected filter is on the MDP_MATRIX table with the synonym M. This filter will not require specifying the table for this table, and should only include the additional WHERE clause used by the filter. Example: WHERE PREDICTION_STATUS=1 Filters including other tables must specify those other tables and join then to MDP_MATRIX. The syntax for the additional tables must begin with a comma, followed by the table name. It is not recommended that tables with a time dimension be used, including SALES_DATA. Care should be taken when defining columns for the filter, as they may exist in multiple tables, causing SQL errors. It is strongly recommended that any column referenced in the filter be prefaced by the table from which it is being retrieved. Example: … ITEMS I WHERE I.ITEM_ID=M.ITEM_ID AND I.ITEM_ID < 100	Global Setting Only

Parameter	Location	Default	Engine Mode*	Details	Tuning
				Example: … T_EP_FRANCHISE T1, T_EP_RETAILER T2 WHERE T1.FRANCHISE_ID=M.FRANCHISE_ID AND T2.RETAILER_ID=M.RETAILER_ID AND T1.FRANCHISE_CODE < 100 AND T2.RETAILER_CLASS > 5	
PopulationFilter	Engine > Shell	Null	Both	Applies a filter to the set of item/location combinations included in a batch profile. The engine will completely ignore combinations that do not meet the filter criteria. The default is no filter. This filter supports columns on the MDP_MATRIX table. Example- CONSUMPTION_DEMAND=1. If the profile is on a General Level then columns from the General Level table can also be referenced.	Global setting only
PROMO_AGGR_LEVEL	Engine > Shell		PE only	**Read-only.** Specifies which level is used as the lowest promotional level of the forecast tree. To specify this parameter, you use the Forecast Tree Editor within the Business Modeler.	Global setting only
PromotionStartDate	Engine > Time		PE only	**Parameter is visible only to owner.** Earliest date for which promotion data can be considered reliable. The Analytical Engine ignores any promotion data before this date. This parameter applies only to combinations that have promotions.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
PromotionSeries	Engine > Proport	null	Globally Defined	see chapter Configuring the Analytical Engine, section Split Forecast by Series	Global Setting only
proport_missingm	Engine > Proport	treated as zero observations	Both**	Specifies how missing dates are treated. Use one of the following values: • treated as zero observations: The missing dates are set equal to zero. That is, suppose that you have three months worth of data as follows: 30, null, 60. If proport_missing equals 0, the average of these three months is calculated as 30 (or [30+0+60]/3) • treated as missing: The missing dates are assumed to have average values. Using the previous example, if proport_missing equals 1, the average of these three months is calculated as 45 (or [30+60]/2). This is mathematically equivalent to assuming that the missing month has average sales (45).	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning

proport_spread	Engine > Proport	receive 0 proportions/global proportions	Both**	Specifies how months that are missing from historical data are filled. Use one of the following values:
				• receive zero proportions: For each missing month, set the proportions equal to 0.
				• receive global proportions: For each missing month, set the proportions equal to glob_prop. In this case, Demantra checks the value of the proport_missing parameter and then does the following:
				  • If proport_missing equals 0, then missing months receive glob_prop*delta.
				  • If proport_missing equals 1, then missing months receive the rolling average (glob_prop).
				• receive 0 proportions/global proportions: For missing months that would have occurred after the first sale for this combination, assign 0 proportions. For months that could not occur in the range of first sale- end of sales, use glob_prop. In this case, for months that could not
					Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
				have been included, Demantra checks the value of the proport_missing parameter and then does the following: • If proport_missing does not equal 1, then missing months receive the rolling average (glob_prop). • If proport_missing equals 1, then missing months receive glob_prop*delta.	

Parameter	Location	Default	Engine Mode*	Details	Tuning
proport_ threshold	Engine > Proport	0	Both**	Specifies how many different months of the year must include data in order for Demantra to calculate proportions for the individual months (P1 - P12, PW1-PW6, etc.). Use any integer from 0 to 12, 24, inclusive. For each combination, the number of unique observable buckets is found (having 3 different observations of January counts as only one month). If not enough months have non-null values, Demantra checks the value of the proport_missing parameter and then does the following: If proport_missing equals 0, then missing months receive glob_prop*delta. If proport_missing equals 1, then missing months receive the rolling average (glob_prop).	Global setting only
ProportParallelJobs	Engine > Proport	1.00	Both**	The number of parallel jobs used when running Proport calculations. This parameter's value should not exceed the number of CPUs on the database server.	Global setting only
ProportRunsInCycle	Engine > Proport	1.00	Both**	The number of groups that the Proport process is broken down into.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
ProportTable Label	Engine > Proport		Both**	The name of the level by which the process is broken down. The total number of members in this level is divided into equally sized groups, and one group is processed each time Proport is run.	Global setting only
Q					
Quantile	Engine > Outlier and Regchange	2.5	Both	**Visible only to owner.** This parameter is used by the validations module of the Analytical Engine, when checking the influence of outliers. It specifies a standard normal percentile for detecting outliers at a prescribed significance level.	Can be tuned by Node
quantity_form	Engine > Data Manipulation	See details.	Both	**Visible only to owner.** Expression that the Analytical Engine uses to compose the historical demand from the sales_data table; the result of this expression is the data that the engine receives as input. This expression should return 0, null, or a numeric quantity for any date. A date with 0 is treated as if there were no sales. A date with null is treated as a missing date; in this case, the system can interpolate a value or just ignore the date. On Oracle, the default is as follows: nvl(pseudo_sale,actual_quantity)*(1 + nvl(demand_fact,0)) + nvl(demand_lift,0)	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
R					
RegimeThreshold	Engine > Outlier and Regchange	5	Both	This parameter is used by the preprocessing module of the Analytical Engine. It specifies the sensitivity of regime change detection. The smaller the value, the more aggressively the engine will detect regime changes. This parameter is used only if regime change is enabled (via the detect_cp parameter).	Can be tuned by node
RemoveResidOutlier		0	Both	Specifies the percentage of residuals (by number) to remove before validating the fit. The residuals are sorted by size and the largest residuals are removed.	Can be tuned by node
ResetForeVals	Engine > Shell	yes	Both	**Visible only to owner.** Specifies whether the engine should clear out previous forecast data before generating the forecast. Use one of the following values: • yes: Demantra clears the previous forecast for all combinations with prediction status equal to 99. (The other combinations are left alone, because the engine will overwrite their forecast anyway.) • no: Demantra does not clear out the previous forecast. This is less ideal but runs more quickly.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
resetmat	Engine > Shell	yes	Both	**Visible only to owner.** Use one of the following values: • yes: Reset loc_node, item_id, and location_id in mdp_matrix. • no: Do not reset these fields.	Global setting only
RunInsertUnits			Both	Specifies the behavior of the INSERT_UNITS procedure, which Demantra calls at the start of an engine run. This procedure makes sure the engine has rows to write into when generating the forecast. This parameter also controls whether Demantra runs the active rolling data profiles when it runs this procedure. Use one of the following values: • 0 means that Demantra does not insert rows and does not execute the rolling data profiles. • 1 means that Demantra insert rows and executes the active data profiles (by running the EXECUTE_PROFILES procedure). • 2 means that Demantra does not insert rows, but does execute the active data profiles.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
RUNMODE			Not applicable	**Read-only; parameter is visible only to owner.** Specifies which version of the Analytical Engine to use. • Use 1 to specify the Promotion Effectiveness version. • Use 0 to run the engine in demand planning mode. This mode will not generate promotional lift. If you use this setting make sure that the LPL is the same as the minimum forecast level.	Global setting only
RunPartialDivider S	Engine> Shell	No - Assign new Branch to every combination participating in forecast	Both	This parameter is used to control whether all combinations are assigned a branch at the beginning of a new engine run. Setting of No (default) will assign a branch ID to all combinations taking part in the engine run. Setting of Yes will assign Branch ID only to nodes that currently do not have a branch ID. Note: Setting value to No can improve engine run performance but may result in unbalanced engine tasks over time.	Global Setting Only

Parameter	Location	Default	Engine Mode*	Details	Tuning
SdeAnalyzeSwitch	Engine > General	yes	Both	Specifies how the Analytical Engine should analyze the sales_data_engine table. Use one of the following values: • yes: Use external logic to analyze this table. See "Reconfiguring the sales_data_engine Table". • no: Analyze the sales_data_engine table as usual.	Global setting only
SdeCreateJoin	Engine > General	no	Both	Specifies whether the Analytical Engine should join sales_data_engine (or its synonym) and mdp_matrix during its run. Use one of the following values: • yes: Join sales_data_engine and mdp_matrix. • no: Do not join these tables. See "Reconfiguring the sales_data_engine Table".	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
SdeCreateSwitch	Engine > General	internal logic	Both	Specifies whether to use external logic to create the sales_data_engine table. Use one of the following values: • use internal logic (0): Create the sales_data_engine table using internal logic. • use external logic (1): Use external logic. If you use this option, you must rewrite the create_process_temp_table, create_object, and drop_object procedures. See "Reconfiguring the sales_data_engine Table". • use external logic done by engine (2). When forecasting on general levels (for example, for service parts forecasting)l, set to external logic engine.	Global setting only
season	Engine > Time	season length	Both	**Read-only.** Season length (52 for weekly systems, 12 for monthly, 7 for daily).	Can be tuned by node
set_rb	Engine > Shell	SET transaction use rollback segment RB1	Both**	**Oracle 8i only; visible only to owner.** Set Rollback Segment command for the database. This is database dependent. See your database documentation.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
ShapeSign	Engine > Data Manipulation		Both	Specifies the signs for the shape causal factors when using them in non-negative regression. Use one of the following values: • 0 means that after the preliminary estimation, the signs are kept as is. • 1 means that after the preliminary estimation, the shape casual factors are made positive. This parameter is ignored if UseNonNegRegr is set to prevent negative coefficients.	Can be tuned by node
ShiftBaseCausals	Engine > Shell	0	PE only	**Parameter is visible only to owner.** Specifies the number of base time buckets by which the baseline causal factors should be shifted; this applies to the causal_factors table. Specify an integer (can be negative). The default setting (0) is recommended.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
ShiftDynPromoDate			PE only	SQL expression that returns the number of days to add to the sales date for any given promotion; typically this is a negative number. If the resulting dates are already in the Inputs table, the Analytical Engine inserts those dates into promotion_data with is_self equal to 0. If this expression is null, then the default promotion dates are used. • If the expression aggregates multiple rows from promotion_data, then be sure to use an aggregate function such as DISTINCT. • Dates are compared to the dates in the Inputs table. If a newly generated date does not match a date in that table, then the date is deleted. • You can apply filters on the resulting dates, via the Promotional Causal Factor window in the Business Modeler.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
ShiftPromoCausals	Engine > Shell	0	PE only	**Parameter is visible only to owner.** This parameter is a global setting that applies to all promotions. You may want to use ShiftDynPromoDate instead, because that gives a greater amount of control. This parameter specifies the number of base time buckets by which the promotional causal factors should be shifted; this applies to the causal factors in the m3_causal_factors table. Specify an integer that can be negative. For example, to make the promotional causal factors active one week after the promotions occur, specify 1 (in a weekly system).	Can be tuned by node
ShiftPromoMaxValue			PE only	Specifies the number of additional future time buckets to bring into history, when shifting promotions to the dates given by ShiftDynPromoDate. By default, the Analytical Engine considers only historical promotions and ignores any future promotions. If you shift promotion dates, that typically means you need to shift promotions that are planned for the very near future. This parameter specifies how many time buckets of the future the Analytical Engine should consider when it shifts the promotion dates.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
start_date	Engine > Time	1-1-1995	Both	First sales date, the start date as it appears in the Inputs table. Can be changed according to the length of history needed for fit estimation and for the proport mechanism. It is strongly recommended this parameter be reset to beginning actual date where history begins. See also the HistoryLength parameter.	Global setting only
start_new_run	Engine > Shell	Yes	Both	Specifies whether to start a new Analytical Engine run or to perform an engine recovery. Internally, the engine records information to indicate its current processing stage. As a result, if the previous engine run did not complete, you can run recovery, and the Analytical Engine will continue from where it was interrupted. Use one of the following values: • yes: Always start a new run, regardless of the status of the last run. • no: Detect whether the previous run was complete and perform a recovery if the previous run did not complete. • prompt: Detect whether the previous run was complete and ask whether to perform a recovery run or a new run.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
StartAverage	Engine > Data Manipulation	-12 for monthly -52 for weekly -7 for daily	PE only	**Promotion Optimization only; parameter is visible only to owner.** Controls the starting date of the time span used to calculating the average baseline forecast. You specify this date relative to last_date. The length of this span of time is controlled by the AverageHorizon parameter. For information on configuring Promotion Optimization, see "Configuring Promotion Optimization for PTP" in the *Oracle Demantra Implementation Guide*.	Global setting only
StdRatio	Engine > Validation	3	Both	This parameter is used by the fit validation module of the Analytical Engine, and it controls the sensitivity of one of the fit validation tests. In this test, the residuals are split into two parts (earlier and later) controlled by TestPeriod. The parameter StdRatio is the maximum allowed ratio of the standard deviation of the later part to the standard deviation of the earlier part. A model is rejected if it fails the test.	Can be tuned by node
T					

Parameter	Location	Default	Engine Mode*	Details	Tuning
TargetTaskSize	Parameters > System Parameters > Engine > Proport	0	Both	Specifies how many MDP_MATRIX combinations the Analytical Engine attempts to assign to each forecasting task. Allocation will be affected by forecast tree branch size. When this parameter is set to 0 (the default value), the system automatically calculates a value for 'TargetTaskSize' depending on the number of engines. Otherwise, the value of 'TargetTaskSize' is used. (Oracle does not recommend changing the default value.) The engine divider uses the value of 'TargetTaskSize' as a system-preferred branch size to create branches that are more equal in size which improves engine performance. The engine divider will try to add as many tasks as possible to an existing branch, up to the limit of 'TargetTaskSize' level 1 combinations, before adding new branches.	Global setting only
test_samp_len	Engine > Validation	6 for monthly data, 26 for weekly 7 for daily	Both	This parameter is used by the fit validation module of the Analytical Engine. It specifies the length of data for forecast validation.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
TestPeriod	Engine > Validation	6 for monthly data, 26 for weekly 7 for daily	Both	This parameter is used by the fit validation module of the Analytical Engine, and it controls the sensitivity of one of the fit validation tests. In this test, the residuals are split into two parts (earlier and later) controlled by TestPeriod. The parameter StdRatio is the maximum allowed ratio of the standard deviation of the later part to the standard deviation of the earlier part. A model is rejected if it fails the test.	Can be tuned by node
TooFew	Engine > General	2	Both	This parameter is used by the preprocessing module of the Analytical Engine. It specifies the minimum number of non-zero data points that a series must have in order for the Analytical Engine to consider it model-feasible. In this test, leading zeros may or may not be considered (depending on the setting of CutTailZeros). Trailing zeros are ignored in either case. Must be 1 or greater. If the series has too few data points, the forecast failure module is run.	Can be tuned by node
top_level	Engine > Shell		Both	**Visible only to owner; read-only.** Indicates the highest level of the forecast tree (the highest fictive level, HFL). This indicates the number of levels that the forecast tree contains.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
TopCoefficientLevel	Engine > Data Manipulation		PE only	**Applies only to Promotion Optimization; parameter is visible only to owner.** Specifies the highest forecast tree level for which the Analytical Engine will calculate coefficients. Use any forecast tree level between BottomCoefficientLevel and the InfluenceRangeLevel, inclusive. For information on configuring Promotion Optimization, see "Configuring Promotion Optimization for PTP" in the *Oracle Demantra Implementation Guide*.	Global setting only
TrendDampPeriod			New	Used during trend detection, this parameter specifies a block of time (as a number of buckets) over which the dampening is applied. The time that contains trend is divided into blocks, as specified by this parameter. For the nth block, the Analytical Engine applies a dampening factor n times. The result is exponential dampening.	Can be tuned by Node
TrendDampStep			New	Used during trend detection, this parameter specifies the dampening factor, which is applied n times to the nth block of time within the trend. The result is exponential dampening. Use a value between 0 and 1, inclusive; smaller values cause dampening to happen more quickly.	Can be tuned by Node

Parameter	Location	Default	Engine Mode*	Details	Tuning
TrendModelForShort			New	Used during trend detection, this parameter specifies which engine model to use in order to generate the trend causal factor. • 1 means use the REGR model. • 2 means use the HOLT model.	Can be tuned by Node
TrendOutlierRatio			New	Used during trend detection, this parameter specifies how to treat outliers during model fit. It specifies a numeric weight to apply to the outliers within the long segment.	Can be tuned by Node

Parameter	Location	Default	Engine Mode*	Details	Tuning
TrendPeriod	Engine > Adjustment		Both	This parameter is used in two parts of the Analytical Engine. The adjustment module uses it as follows: • If EnableAdjustment is yes (1), then TrendPeriod specifies how far back in history the trend is measured for adjustment. • If zero, then no adjustment is performed. Enabling adjustment is not recommended, unless it is known that a change in trend happened recently, which is likely to be missed by the models. This parameter is also used by trend detection as discussed in "The Forecasting Process., page 9-6". If you have disabled negative regression (via UseNonNegRegr), then it is difficult for the Analytical Engine to detect downward trends. In such cases, you should enable trend detection.	Can be tuned by node
TrendPreEstimation			New	Specifies whether to perform trend detection as described in "The Forecasting Process".	Can be tuned by Node
TrendShortRatio			New	Used during trend detection, this parameter specifies how to treat outliers during model fit. It specifies a numeric weight to apply to the outliers within the short segment.	Can be tuned by Node
U					

Parameter	Location	Default	Engine Mode*	Details	Tuning
UpliftThresholdMethod	Engine > Validation	1	PE only	Specifies how to determine the uplift threshold: • 0 means use absolute values • 1 means use percent of baseline	Global setting only
UpliftThresholdValue	Engine > Validation	.5	PE only	Uplift and cannibalization values lower than this threshold are automatically set to null. This number must be greater than 0.	Global setting only
UpperUpliftBound	Engine > Validation	20	PE only	**Visible only to owner,** Specifies the upper allowed limit for uplifts, as a proportion of baseline. For each forecasting model, the Analytical Engine calculates the lift for each node of the forecast tree. For any given node and model, if the absolute value of the uplift is greater than this limit, then that model is not used for this node. The engine discards a model (for a given forecast node) in either of two cases: • If any uplift exceeds the bound given by the UpperUpliftBound parameter. • If the model generates too many exceptional uplifts (as specified by the LowerUpliftBound and AllowableExceptionsparameters).	Can be tuned by Node

Parameter	Location	Default	Engine Mode*	Details	Tuning
UpTime	Engine > Data Manipulation	nvl(sum(nvl(UP_TIME,1)), 1)	Both	This parameter is used by the preprocessing module of the Analytical Engine. It is used to flag whether each date in sales_data should be considered a sales date or not. Use an SQL expression that returns one of the following values: • 0 (to indicate a no-sales date) • 1 (to indicate a date on which sales could theoretically happen)	Global setting only
UpTrend	Engine > Adjustment	0.2	Both	This parameter is used by the adjustment module of the Analytical Engine, if that module is enabled (via EnableAdjustment). It controls forecast adjustment for upward trend. Specifically, it represents the amount the forecast is rotated to align with recent trend in data. Use a value from 0 to 1, inclusive. Enabling adjustment is not recommended, unless it is known that a change in trend happened recently, which is likely to be missed by the models.	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
UseBusinessFilter	Engine > Data Manipulation	no	Both	Specifies whether the Analytical Engine distinguishes business and non-business days. Use one of the following values: • yes: The Analytical Engine uses only business days (as indicated by business_day_filter series in the Inputs table). • no: The Analytical Engine uses all days.	Global setting only
UseEnvelope					

Parameter	Location	Default	Engine Mode*	Details	Tuning
UseExternalSDUpdate	Engine > Shell	no	Both	**Visible only to owner.** Specifies how to update the sales_data table with the current forecast. Use one of the following values: yes: Use an external procedure (create_process_temp_table. no: Use an internal dynamic procedure. The create_process_temp_table procedure is a template that creates the dynamic stored procedure that will be executed by the engine for the update. By default, this procedure creates the same SP that as the engine creates, but this can be overridden. The interface to this SP is as follows: • is_proc_name (VARCHAR2) specifies the name of the dynamic SP that the engine will execute. • is_tmp_tbl (VARCHAR2) specifies the temptable name. • is_fore_col (VARCHAR2) specifies the column name in sales_data that will be updated with the new forecast. • is_last_date (VARCHAR2) specifies a date to update mdp_matrix with.	Global setting only

Parameter	Location	Default	Engine Mode*	Details	Tuning
usemodelspernode	Engine > Data Manipulation		Both	Specifies whether you can specify the forecasting models to use for specific nodes in the forecast tree, via the File > Analytics menu option in Promotion Effectiveness.	Global setting only
UseNonNegRegr	Engine > Validation	yes	Both	**Visible only to owner.** Specifies whether to use non negative constraint estimation for all regression-based engine models. Use one of the following values: • yes: The coefficients are prevented from being negative. • no	Can be tuned by node
UseParamsPerNode	Engine > Data Manipulation		Both	Specifies whether you can specify the engine parameters to use for specific nodes in the forecast tree, via the File > Analytics menu option in Promotion Effectiveness.	Global setting only
UseWeightedRegression		0	Both	Specifies whether the Analytical Engine applies a weight to each observation when fitting each model. • If this parameter is set to 1 (yes), the OBS_ERROR_STD field (in sales_data) specifies the weights for each observation. • If this parameter is 0 (no), that field is ignored.	Can be tuned by Node
W					

Parameter	Location	Default	Engine Mode*	Details	Tuning
WriteIntermediateResults	Engine > Shell	no	Both	**Applies only to the desktop products; parameter is visible only to owner.** Specifies whether to enable the advanced analytics function, which is available only on the desktop. Use one of the following values: • yes: Retain intermediate results (coefficients for causal factors) to enable advanced analytics. The results are written to the INTERM_RESULTS table. This information includes the coefficients for each model, and the weight of each model in the forecast. **Warning:** The Analytical Engine will run much more slowly. • no	Can be tuned by node

Parameter	Location	Default	Engine Mode*	Details	Tuning
WriteMissing DatesUplift	Engine > Shell	no	PE only	**Parameter is visible only to owner.**Specifies whether to write uplifts for dates that are missing from sales_data. Use one of the following values: • yes: The Analytical Engine writes uplifts for any dates where it calculates them, even if no sales occurred. However, the uplifts will add up to the total uplift calculated by the engine. • no: The Analytical Engine writes uplifts only for dates that have sales. This means that the uplifts will not necessarily add up to the total uplift.	Global setting only

* **PE only** means PE mode only; **Both** means both PE and DP modes. ** This parameter is not used directly by the engine and thus is available for use with either engine mode.

Database Parameters for Tuning Engine Performance

The following parameters provide more advanced controls of how Demantra executes specific engine processes and allow you to tune the engine based on your implementation's unique characteristics. These parameters can be set in the INIT_PARAMS_% tables.

> **Warning:** Using these parameters to improve engine performance depends almost entirely upon the expertise of the implementer. System issues may result if they are not used correctly. Oracle does not recommend using these parameters unless you have been trained and have previous experience tuning the Analytical Engine

Parameter	Details

DBHintBranchDivision	This parameter applies hints to Engine DB statements that divide MDP_MATRIX combinations among forecast engine tasks. When the hint references to table MDP_MATRIX, use #DYNTABLE1# instead. The engine will automatically replace with MDP_MATRIX. For example, to run four parallel processes when executing branch division, set this parameter to + parallel(#DYNTABLE1# 4).
DBHintInitialForeCleanup	This parameter applies hints to the EngineManager's cleanup process before the actual run is started. For referring to the table, use the dynamic table token #DYNTABLE1# . For example, to run four parallel processes, you would set this parameter to: DBHintInitialForeCleanup = '+ parallel(#DYNTABLE1#,4)'
DBHintAggriSQL	This parameter applies hints to Engine DB statements that aggregate data from the SALES_DATA table. When the hint references to table SALES_DATA, use #DYNTABLE1# instead. The engine will automatically replace with SALES_DATA. For example, to run four parallel processes when executing aggregation statements, set this parameter to + parallel(#DYNTABLE1# 4).
DBHintLoadActual	This parameter applies hints to the Engine DB statement that retrieves the lowest level historical data during an Analytical Engine run. When referencing a table use #DYNTABLE1# or #DYNTABLE2# to designate the table. The Engine automatically replaces dynamic strings with the relevant table name or alias. #DYNTABLE1# references SALES_DATA and #DYNTABLE2# references MDP_MATRIX. For example, to run four parallel processes on MDP_MATRIX and six parallel processes on SALES_DATA when querying lowest level information, set this parameter to + parallel(#DYNTABLE1# 6) parallel(#DYNTABLE2# 4).

DBHintCreatePDE	This parameter applies hints to the Engine DB statements that create the temporary promotion data engine tables during an Analytical Engine run. When referencing a table, use #DYNTABLE1#, #DYNTABLE2# or #DYNTABLE3# to designate the table. The Engine automatically replaces dynamic strings with the relevant table name or alias. Use #DYNTABLE1# to reference MDP_MATRIX, #DYNTABLE2# to reference PROMOTION and #DYNTABLE3# to reference PROMOTION_DATA. For example, to run four parallel processes on MDP_MATRIX and six parallel processes on PROMOTION_DATA when creating the PDE table, set this parameter to + parallel(#DYNTABLE1# 4) parallel(#DYNTABLE3# 6).

11

Theoretical Engine Models

Important: The Demantra Local Application replaces Collaborator Workbench. You may see both names in this text.

This chapter contains reference information for the theoretical models that the Analytical Engine uses.

This chapter covers the following topics:

- Introduction
- Flags on Causal Factors
- ARIX
- ARLOGISTIC
- ARX
- BWINT
- CMREGR
- DMULT
- ELOG
- FCROST
- HOLT
- ICMREGR
- IREGR
- LOG
- LOGISTIC
- Moving Average
- MRIDGE

- NAIVE
- REGR

Introduction

Note: Oracle provides two different modes for the Analytical Engine:

- In PE mode, the engine is suitable for use with Promotion Effectiveness.
- In DP mode, the engine is suitable for use in demand planning applications.

For each model, this chapter indicates which engine modes that model can be used with.

Flags on Causal Factors

You use the Business Modeler to apply the following flags to the causal factors; see "Configuring Global and Local Causal Factors" and "Configuring Promotional Causal Factors":

Flag*	Meaning
short	For use by the short models (BWINT, IREGR, LOGREGR, LOGISTIC, and REGR). These models use all causal factors that they are given.
long	For use by the long models (ARLOGISTIC, CMREGR, ELOG, ICMREGR, and MRIDGE). These models examine all the causal factors they are given, but choose the ones that give the best results.
non-seasonal	For use by the non -seasonal models (ARIX and ARX). The only causal factors that should be flagged as non-seasonal are ones that are not a predictable function of time. For example, price varies with time, but randomly, so price should be flagged as non-seasonal.
multiplicative group 1	For use only by the DMULT model. If you are using this model, each causal factor should use one of these flags.
multiplicative group 2	See "DMULT".

Flag*	Meaning

*Name of flag as displayed in the Causal Factors screen or in the Promotional Causal Factors screen.

Models not listed here use other mechanisms to choose their causal factors or do not use causal factors at all.

ARIX

ARIX includes integrated auto-regression terms at lag 1 and an unknown seasonal lag *k*, and linear causal factors.

The value of *k* is chosen from set of possible seasonal indexes to produce the best fit. Causal factors include the constant and events (without seasonal causal factors and without time).

Availability

ARIX can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes*
DP mode	Yes

*The ARIX model is never used on promotional nodes. See "Summary of the Forecasting Process".

Causal Factors Used by This Model

ARIX uses the non-seasonal causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
Possible Season*	**For daily data:** 2, 3, 4, 5, 6, 7, 14, 30, 31, 90, 91, 92, 182, 365 **For weekly data:** 2, 4, 5, 13, 14, 26, 52 **For monthly data:** 3, 6, 12.	A vector of possible seasonal patterns of the series. The parameter is of type vector (other parameters are defined as double in PARAM_TYPE column), with an increasing index (PARAM_INDEX) for each new PARAM_VALUE.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

The ARIX parameters also apply to the ARX model.

ARLOGISTIC

ARLOGISTIC is an extension of the LOGISTIC model and includes auto-regression and logistic regression terms.

Availability

ARLOGISTIC can be used with the following engine modes:

Engine Mode	Supported?
PE mode	No (disable model if using this mode)
DP mode	Yes

Causal Factors Used by This Model

ARLOGISTIC uses the long causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

ARLOGISTIC uses the same parameters as LOGISTIC; see "Parameters Used by This Model".

ARX

This model includes auto-regression terms at lag 1 and an unknown seasonal lag k, and linear causal factors. The value of k is chosen from set of possible seasonal indexes to produce the best fit. Causal factors include the constant and events (without seasonal causal factors and without time).

Availability

ARX can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes*
DP mode	Yes

*The ARX model is never used on promotional nodes. See "Summary of the Forecasting Process".

Causal Factors Used by This Model

ARX uses the non-seasonal causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

ARX uses the same parameters as ARIX; see "ARIX".

BWINT

BWINT (the Multiplicative Regression-Winters model) runs multiplicative regression on the causal factors, then exponentially smooths the resulting residuals in HOLT manner and then runs multiple regression of the smoothed residuals. BWINT models trend, seasonality and causality.

Availability

BWINT can be used with the following engine modes:

Engine Mode	Supported?
PE mode	No (disable model if using this mode)
DP mode	Yes

Causal Factors Used by This Model

BWINT uses the short causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
Alpha*	0.1	The manually set level renovation coefficient, valid only when OptimizedBwint* = 0.
Gamma*	0.3	The manually set trend renovation coefficient, valid only when OptimizedBwint* = 0.
OptimizedAlphaIter*	3	The number of values on the Alpha grid for parameters optimization.
OptimizedBwint*	0	Specifies whether the parameter values (Alpha & Gamma) of the Holt procedure used here are to be optimized (1) or preset (0).
OptimizedGammaIter*	10	The number of values on the Gamma grid for parameters optimization.
Phi*	0.9	The trend damping coefficient, always set manually.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

CMREGR

CMREGR (the Markov Chain Monte-Carlo model) fits to data an assortment of linear functions of the form: Series= Causals*Coeff + Resid.

Where:

- Causals are various subsets of causal factors, chosen by a random process from all possible combinations of factors.

The first set of causal factors consists of a collection of factors along with the lagged time series. Then, for a given length, a chain of that length is generated, and that path of that Markov chain is traveled. The states of the chain are subsets of factors, the transition probabilities for neighboring states are based on the ratio of BICs (Bayesian Information Criteria) and are zero for non- neighboring states. Neighboring states are states that differ only by one member. At each pass, a new factor is chosen randomly. If the current model does not contain this factor, it joins, with the calculated transition probability, the group to form the next model. Thus, the greater the improvement in the model (as measured by BIC), the more probability has the model to be employed. If the current model already contains this factor, then, with the calculated transition probability, it leaves the group.

Also, a special causal factor Lag is used; this is merely the original series lagged back by one time period. When the procedure finds this causal factor useful for modeling, the meaning is that there is a significant autoregressive component in the data, which indicates the presence of random trends. If the influence of Lag is dominant over other factors, which is indicated by a large Lag coefficient, the fit will inhere the lagging effect and when plotted on the same graph as the original series, will seem to "echo" previous observations. This means that the model was unable to pick up any systematic behavior in the series, and the best it can do is to highly correlate fitted values with lagged data.

Availability

CMREGR can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

CMREGR uses the long causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
Reset_Seed*	1	Specifies whether to reset the seed for random numbers generation at each run or simulation. If the seed is not reset, there will be different results for each run; also the simulation results will differ from batch results. 1= reset_seed; 0 = do not reset seed. Theoretically the model assumes that the seed is not reset.
ChainLength*	500	Number of models considered for averaging.
Need_Lag*		Specifies whether to use the Lag as a causal factor. Lag - the previous actual observation explains the next one.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.
UseEnvelope	no	Specifies whether Demantra will use the envelope function described in "Causal Factor Testing (Envelope Function)".
ENVELOPE_RESET_SEED*	0	Specifies whether to reset the randomization seed for the envelope function, which evaluates different sets of causal factors for different engine models.
ENVELOPE_CHAIN_LENGTH*	50	Specifies the number of variations of causal factors to try, for each model.
BestOrMix*	0	Specifies whether to use the best set of causal factors (1) or to use a mix of the causal factors (0). The default is 0.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

DMULT

DMULT, the Multiplicative Multi-Seasonal Regression model, divides causal factors into two groups and combines them in a multiplicative linear function of the following form:

*(sum of values in causal factor group 1) * (sum of values in causal factor group 2)*

This function can be used, for example, to combine daily and monthly seasonality.

Availability

DMULT can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

When you define causal factors and promotional causal factors, the Causal Factors screen and the Promotional Causal Factors screen enable you to place each factor into multiplicative group 1 or multiplicative group 2.

These options correspond to the DAILY_VAL (multiplicative group 1) and MONTHLY_VAL (multiplicative group 2) columns in the causal_factors and the promotional_causal_factors tables.

Typically, one group contains daily causal factors such as the days of the week D1,D2,..,D7. The other group contains the remaining causal factors. Each group should include at least one causal factor, and each causal factor should be in only one group.

Parameters Used by This Model

Parameter	Default	Description
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

Parameter	Default	Description
MAX_ITERATIONS*	3	Specifies the maximum number of iterations used by this model. This parameter must be a whole number greater than or equal to 3. If it less than 3, the Analytical Engine uses the value 3.
SET2_COEFF_INI*	0	Specifies the initial values for the coefficients in multiplicative group 2. The default is 0, which means that the initial values for these is zero, except for the coefficient for the constant causal factor.

* This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

ELOG

ELOG (the Logarithmic CMREGR model) performs the CMREGR procedure on the log-transformed time series.

As with the CMREGR model, this model uses a special causal factor (Lag); see "CMREGR".

Availability

ELOG can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

ELOG uses the long causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
ChainLength*	500	Length of the generated Markov Chain, that is number of models considered for averaging.
need_lag*		Specifies whether to use the Lag as a causal factor. Lag - the previous actual observation explains the next one.
reset_seed*	1	Specifies whether to reset the seed for random numbers generation at each run or simulation. If the seed is not reset, there will be different results for each run; also the simulation results will differ from batch results. 1= reset_seed; 0 = do not reset seed. Theoretically the model assumes that the seed is not reset.
LogCorrection	1	Specifies whether to use (1) or not (0) the correct form of the expectation of a lognormal variable.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.
UseEnvelope	no	Specifies whether Demantra will use the envelope function described in "Causal Factor Testing (Envelope Function)".
ENVELOPE_RESET_SEED*	0	Specifies whether to reset the randomization seed for the envelope function, which evaluates different sets of causal factors for different engine models.
ENVELOPE_CHAIN_LENGTH*	50	Specifies the number of variations of causal factors to try, for each model.
BestOrMix*	0	Specifies whether to use the best set of causal factors (0) or to use a mix of the causal factors (1).

Parameter	Default	Description
*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.		

FCROST

FCROST (the Croston Model for Intermittent Demand) is useful for intermittent demand, which can be viewed as the demand by a distributor that supplies the product to end customers. What is visible to the demand planner is the bulk demand by the distributor, while the periodic demand of retailers is unknown. Thus, the quantities most probably reflect replenishment orders, rather than demand. Visually the data consists of peaks of random height with random intervals between the peaks.

This model is useful for data involving substantial number of zeros, and is particularly relevant for forecasting demand of slow moving parts. The model utilizes the Holt procedure for forecasting both quantities and inter-event times.

Availability

FCROST can be used with the following engine modes:

Engine Mode	Supported?
PE mode	No (disable model if using this mode)
DP mode	Yes

Causal Factors Used by This Model

None.

Parameters Used by This Model

Parameter	Default	Description
AlphaQ*	0.1	Level innovation coefficient for quantities, manually set.
AlphaT *	0.1	Level innovation coefficient for inter-event times, always manually set.

Parameter	Default	Description
GammaQ*	0.3	Trend innovation coefficient for quantities, manually set.
GammaT*	0.3	Trend innovation coefficient for inter-event times, always manually set.
OptimizedAlphaIter*	3	The number of values on the Alpha grid for parameter optimization.
OptimizedFcrost*	0	For forecasting the inter-event times only. Parameter specifies whether the parameter values (AlphaQ & GammaQ) of the quantities-forecasting Holt procedure used here are to be optimized (1) or preset (0). For forecasting the inter-event times only,
OptimizedGammaIter*	10	The number of values on the Gamma grid for parameter optimization.
Phi*	0.9	Trend damping coefficient for inter-event times, always manually set.
PhiQ*	0.9	Trend damping coefficient for quantities, always manually set.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

HOLT

HOLT (the Double Exponential Smoothing model) provides realization for the Holt damped two-parameter exponential smoothing algorithm. The forecast is a projection of the current level estimate shifted by damped trend estimate. The level estimates are computed recursively from data as weighted averages of the current series value and the value of the previous one-step-ahead forecast. The trend (change of level) estimates are computed as weighted averages of the currently predicted level change and

damped previously predicted trend. The weights and the damping coefficient are either user-supplied or can be optimized. If the optimization of parameters is chosen, they will be set so that the MAPE (Mean Square Percentage Error) is minimized.

The HOLT model is suitable for modeling time series with a slowly changing linear trend. It is usually used only to model short series (for example, 52 or fewer data points for a weekly system).

Availability

HOLT can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes*
DP mode	Yes

*The HOLT model is used on promotional nodes only if no other models can be used. See "Summary of the Forecasting Process".

Causal Factors Used by This Model

None.

Parameters Used by This Model

Parameter	Default	Description
Alpha*	0.1	The manually set level renovation coefficient, valid only when OptimizedHolt* = 0.
Gamma*	0.3	The manually set trend renovation coefficient, valid only when OptimizedHolt* = 0.
OptimizedAlphaIter*	3	The number of values on the Alpha grid for parameters optimization.
OptimizedGammaIter*	10	The number of values on the Gamma grid (default) for parameters optimization.
OptimizedHolt*	0	Specifies whether the parameter values (Alpha & Gamma) are to be optimized (1) or preset (0).
Phi*	0.9	The trend damping coefficient, always set manually.

Parameter	Default	Description
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

ICMREGR

ICMREGR (the Intermittent CMREGR model) is an extension of both CMREGR and IREGR models.

Availability

ICMREGR can be used with the following engine modes:

Engine Mode	Supported?
PE mode	No (disable model if using this mode)
DP mode	Yes

Causal Factors Used by This Model

ICMREGR uses the long causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
ChainLength*	500	Length of the generated Markov Chain, that is the number of models considered for averaging.
need_lag*		Specifies whether to use the Lag as a causal factor. Lag - the previous actual observation explains the next one.

Parameter	Default	Description
reset_seed*	1	Specifies whether to reset the seed for random numbers generation at each run or simulation. If the seed is not reset, there will be different results for each run; also the simulation results will differ from batch results. 1= reset_seed; 0 = do not reset seed. Theoretically the model assumes that the seed is not reset.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

IREGR

IREGR (the Intermittent Regression model) is useful because the Croston model fails to consider the obvious interdependency between quantities and times between occurrences of demands in intermittent series. Moreover, due to the nature of the Holt model used by Croston, causalities and seasonality are not modeled. IREGR spreads the data into a continuous series and fits to it a regression model with unequal variances. The resulting fit and forecast may be lumped back to form spikes, after being processed by the Bayesian blending procedure.

Availability

IREGR can be used with the following engine modes:

Engine Mode	Supported?
PE mode	No (disable model if using this mode)
DP mode	Yes

Causal Factors Used by This Model

IREGR uses the short causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

LOG

LOG (the Multiple Logarithmic Regression model) performs a logarithmic regression. Using logarithms is often a good way to find linear relationships in non-linear data.

This model fits to data a linear function of the form:

ln(Series+ones*Shift) = Causals*Coeff + Resid

Where:

- Resid is the vector of residuals.
- ones is a column vector of ones.
- Shift is a calculated value to shift the series away from non-positive values, before the logarithmic transformation.

Forecast values are obtained by back-transforming the projected regression, while considering the theoretical form of the expectation of a log-normal random variable

Availability

LOG can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

LOG uses the short causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
LogCorrection	1	Specifies whether to use (1) or not (0) the correct form of the expectation of a lognormal variable.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.
UseEnvelope	no	Specifies whether Demantra will use the envelope function described in "Causal Factor Testing (Envelope Function)".
ENVELOPE_RESET_SEED*	0	Specifies whether to reset the randomization seed for the envelope function, which evaluates different sets of causal factors for different engine models.
ENVELOPE_CHAIN_LENGTH*	50	Specifies the number of variations of causal factors to try, for each model.
BestOrMix*	0	Specifies whether to use the best set of causal factors (0) or to use a mix of the causal factors (1).

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

LOGISTIC

LOGISTIC runs logistic regression on the causal factors.

Availability

LOGISTIC can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

LOGISTIC uses the short causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

The LOGISTIC parameters also apply to the ARLOGISTIC model.

Parameter	Default	Description
Potential*	1.5	Specifies the upper bound of market effort effect, as a multiple of maximum historical sales.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

Moving Average

The Moving Average model considers the most recent time buckets, computes the average, and uses that for the forecast, resulting in a flat line. This forecast is generally suitable only in the near future.

This model is provided as a possible substitute for the NAIVE model, for use when all other models have failed. It does not generally interact well with other models and so is recommended only for use if no other forecast models have worked.

See "Forecast Failure", and also see "NAIVE".

Availability

The Moving Average model can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes (no lift is generated, however)
DP mode	Yes

Causal Factors Used by This Model

None.

Parameters Used by This Model

Parameter	Default	Description
NaiveEnable		Specifies what to do at the highest forecast level, upon failure of all models. • no (0): Do not enable either NAIVE or Moving Average models. Do not generate a forecast. • yes (1): Enable use of the NAIVE model. • 2 or higher: Enable use of the Moving Average model. In this case, the setting of NaiveEnable specifies the number of recent time buckets to use in calculating the moving average.

MRIDGE

MRIDGE (the Modified Ridge Regression model) produces regression coefficients of moderate magnitude, thus assuring that lifts associated with events are of moderate size. This is equivalent to imposing a set of constraints on the coefficients in a spherical region centered at zero. In the literature, this model is of the shrinkage family.

Availability

MRIDGE can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

MRIDGE uses the long causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
RIDGEK*	1	The larger the value of RIDGEK, the more shrinkage occurs. When RIDGEK=0, the model is equivalent to REGR.
METRIC NORM*	2	Chooses the norm for scaling the input causal factors.
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.
UseEnvelope	no	Specifies whether Demantra will use the envelope function described in "Causal Factor Testing (Envelope Function)".
BestOrMix*	0	Specifies whether to use the best set of causal factors (0) or to use a mix of the causal factors (1).

*This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

NAIVE

The NAIVE model is used only at the highest forecast level, and is used only if all other

models (including HOLT) have failed. See "Forecast Failure", and also see "Moving Average".

It uses a simple averaging procedure.

Availability

NAIVE can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes (no lift is generated, however)
DP mode	Yes

Causal Factors Used by This Model

None.

Parameters Used by This Model

Parameter	Default	Description
NaiveEnable		Specifies what to do at the highest forecast level, upon failure of all models. no (0): Do not enable either NAIVE or Moving Average models. Do not generate a forecast. yes (1): Enable use of the NAIVE model. 2 or higher: Enable use of the Moving Average model. In this case, the setting of NaiveEnable specifies the number of recent time buckets to use in calculating the moving average.
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.

Note: When generating naive forecast at the highest forecast level, one of two methods are used. If Holt was not attempted for this node, a simplified version of the Holt model will be used and the combination will be marked with the letter T. If Holt was previously attempted a moving average based model is used instead and the node is marked with N for Naive.

REGR

REGR (the Multiple Regression model) fits to data a linear function of the form:

Series = Causals * Coeff + Resid

Where:

- Causals is a matrix with the independent variables (causal factors) as its columns.
- Coeff is a column vector of regression coefficient.
- Resid are the (additive) residuals (errors).

Using this additive model, we are assuming that a linear relationship exists. The dependent variable is linearly related to each of the independent variables.

The regression parameters estimates are obtained by using the method of least square error.

Regression coefficients that are not statistically significant are identified by special tests and assigned the value 0.

> **Note:** All regression-based models use REGR implicitly.

Availability

REGR can be used with the following engine modes:

Engine Mode	Supported?
PE mode	Yes
DP mode	Yes

Causal Factors Used by This Model

REGR uses the short causal factors; see "Flags on Causal Factors".

Parameters Used by This Model

Parameter	Default	Description
UseNonNegRegr	no	Specifies whether to constrain the regression coefficients to nonnegative values, within the core least squares estimation.

Parameter	Default	Description
AllowNegative	no	Specifies whether negative values of fit and forecast are allowed. If negative values are not allowed, then any non-positive fitted and forecasted values are set to zero.
UseEnvelope	no	Specifies whether Demantra will use the envelope function described in "Causal Factor Testing (Envelope Function)".
ENVELOPE_RESET_SEED*	0	Specifies whether to reset the randomization seed for the envelope function, which evaluates different sets of causal factors for different engine models.
ENVELOPE_CHAIN_LENGTH*	50	Specifies the number of variations of causal factors to try, for each model.
BestOrMix*	0	Specifies whether to use the best set of causal factors (0) or to use a mix of the causal factors (1).

* This parameter is model-specific and is not displayed in the Business Modeler; see the Parameters table.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z