



Accelerate Waterfall Optimization (POAI)

December 2022

Accelerate Waterfall Optimization (POAI)

The POAI Accelerator provides a fast implementation of a standard waterfall simulation.


In this section:

- [POAI Accelerator Overview](#)
- [POAI Accelerator Deployment](#)
- [POAI Accelerator Run](#)
- [POAI Accelerator Technical Content](#)
- [POAI Accelerator Results](#)

To understand the optimization results and to query them from outside of the Optimization Engine module, please refer to [Optimization Results](#).

POAI Accelerator Overview

POAI Accelerator deploys a standard Optimization Engine model that optimizes elements of a waterfall.

 This Accelerator is a proof of concept meant only **waterfall optimization** use cases.

Optimization Engine can do other things, of course, but this is the main use case and any other custom requirements cannot be achieved via this Accelerator.

To be as fast and generic as possible, a standardized waterfall is built into the Accelerator and is designed to cover most of the customer needs. This waterfall looks like this:

pricefx

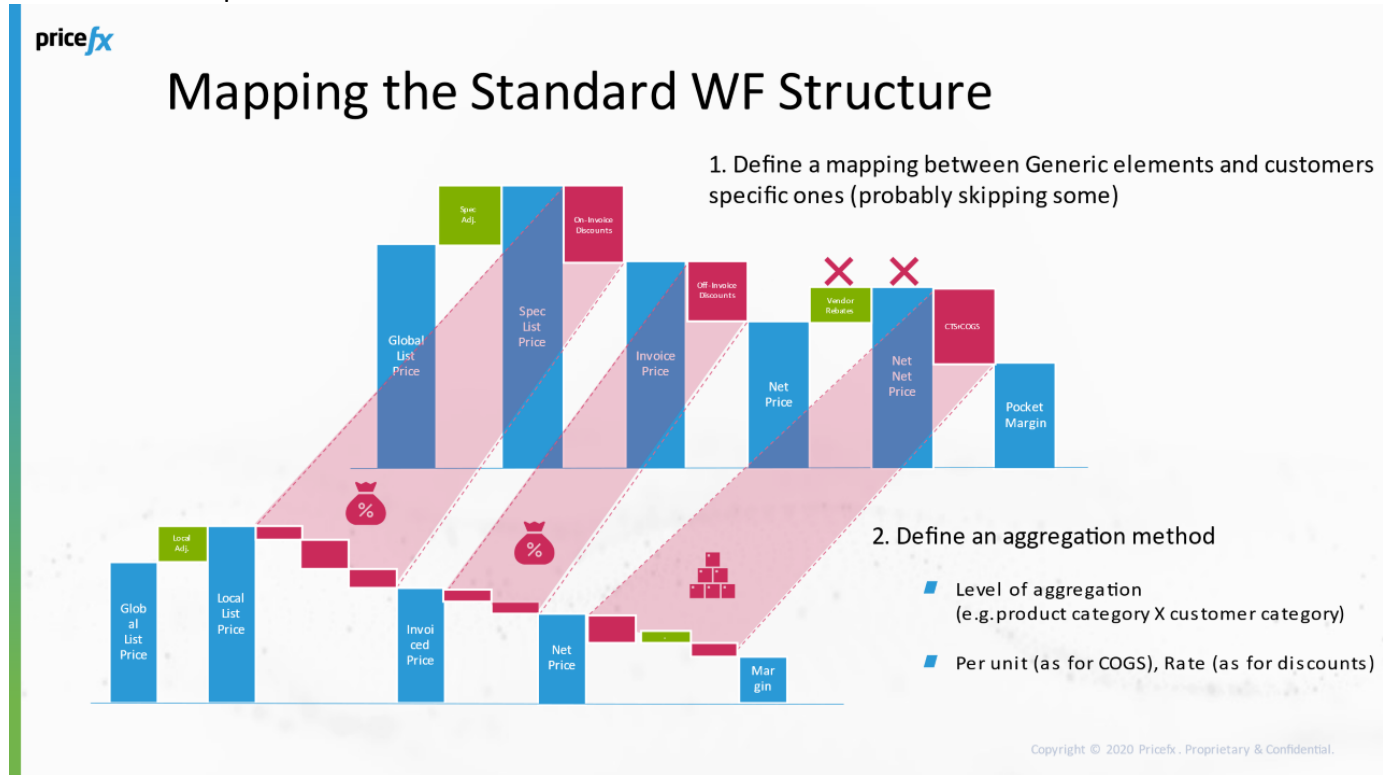
Standard WF Structure



Built down to the level of Product X Customers

A key part of the setup is mapping the actual customer waterfall columns to this standard one. This could mean that some of the elements of the customer waterfall will be aggregated within a single element of the standard waterfall and that some elements may not be used at all.

Illustrative example:



As a result:

- Customers need to be aware of the fact that the **results will not reflect their exact waterfall**.
- We can standardize most things - mainly the way we consume results (Optimization Realization Dashboards, Price Setting, Agreements & Promotions, etc.).

POAI Accelerator Deployment

POAI Accelerator deploys a standard POAI model that optimizes the elements of a standard waterfall.

In this section, you will find all information to deploy and run a standard waterfall optimization. More details on data mapping, elasticity model creation, and optimization results evaluation are given in the children pages.

- [Prerequisites](#)
- [Deployment](#)

Prerequisites

- **Transactions Datamart** - The Accelerator deployment requires fields mapping to define the price waterfall levels. You need a Transactions Datamart defining all the waterfall fields. For details see [POAI Accelerator Data Mapping](#).

- **Segmentation Model** - To create and run an optimization model deployed via an Accelerator, you need a Segmentation Model on the product_group and customer_group fields you want to consider during the optimization. This segmentation will provide the elasticity values to the optimization model. For details see [Elasticity Model for POAI Accelerator](#).
- **Image** - To run an optimization model, the image `registry.pricefx.eu/engineering/pricefx-optimization-engine` has to be enabled in the partition. Please [create a helpdesk ticket](#) if the partition does not have it enabled yet. You will be asked for the number of CPUs and memory size to run the model. In terms of CPUs, 2 is enough; you can ask for 4 if you need to make the jobs faster. In terms of memory, you can start with 4 Gi and ask for increase if it is not enough.

Deployment

1. Access PlatformManager at <https://platform.pricefx.com/> and log in with your account or using 0365.
2. Go to **Marketplace > Templates Management**.
3. Find POAI Accelerator.
4. In the row menu select **Show Detail**:

The screenshot shows the 'Templates Management' interface. On the left, a sidebar contains navigation options: 'Selected account', 'Integration Templates', 'Accelerators', 'Accelerator Packages', 'Non Approved', 'Templates Management' (highlighted with a red box), 'Deployed Templates', and 'Failed Deployments'. The main content area is titled 'Templates Management' and features a search bar with 'poai' entered. Below the search bar, a table lists templates. The first row is 'POAI Accelerator', which is highlighted with a red box. A context menu is open for this row, with 'Show Detail' highlighted by a red box. Other menu options are 'Edit' and 'Permissions'. The second row is 'POAI Library'.

5. Choose a valid version you want to deploy and click the deployment icon. Generally, the best version is the last valid one, except when the partition runs too old version of Pricefx.

net.pricefx.accelerators:poai-accelerator

Version	Release Date	Validation Date	Published	Min. Integration...	Min. Partition V...	Valid	
0.1-SNAPSHOT	30/03/2021	17/12/2021 15:42	<input type="radio"/> No	[,1.1.18.1)	1.0.0	⊙	🔗
0.1.0	27/01/2022	27/01/2022 15:12	<input type="radio"/> No	[,1.1.18.1)	1.0.0	⊗ ↓	🔗
0.2-SNAPSHOT	22/09/2021	01/02/2022 14:00	<input type="radio"/> No	[,1.1.18.1)	1.0.0	⊙	🔗
Dev1.0	31/08/2021	14/09/2021 14:44	<input type="radio"/> No	[,1.1.18.1)	1.0.0	⊙	🔗
DEVELOP	26/01/2022	09/02/2022 08:...	<input type="radio"/> No	[,1.1.18.1)	1.0.0	⊙	🔗

6. Select your partition from the drop-down menu and click **Continue**.
7. Set up Datamart mapping.

Settings

Source Name

Product

Product Label

Product Group

Product Group Label

Customer

Customer Label

Customer Group

Customer Group Label

- Some rules apply in the mapping:
 - Labels are not mandatory. If not filled, a related name field will be used.
 - Product Group and Customer Group fields must *match the segmentation model* used by the optimization.
 - Numerical values must be *extended*.
 - More details, particularly if your waterfall does not match the standard Accelerator one, are in [P OAI Accelerator Data Mapping](#).
8. Click Continue and wait until the deployment is complete.

Congratulations!

Your Accelerator / Accelerator Package was successfully deployed. Continue to see the result.

Finish

Go to partition



POAI Accelerator Data Mapping

This section details how to create the data structure needed if you want to use the POAI Accelerator and how to configure it during the deployment.

- [Objectives of Data Mapping](#)
- [Standardized Waterfall](#)
- [What to Map and How](#)
 - [Transactions Datamart Prerequisites](#)
 - [Mapping during the Accelerator Deployment](#)
 - [Dealing with Missing Fields](#)

Objectives of Data Mapping

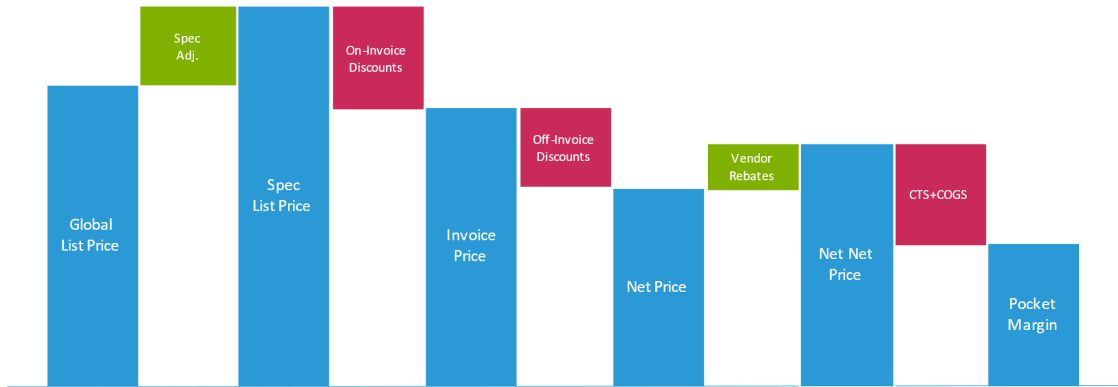
By default, the Accelerator generates an “aggregation” table containing data in a format understood out-of-the-box by the deployed logics. For this, the Accelerator needs to have the raw transactions lines available in a Datamart. Then, the Accelerator will ask for a mapping between the fields of the transactions Datamart and the ones that will be used in the Accelerator.

This approach ensures that the expected fields are correctly aggregated and that their names are compatible with the deployed logics.

Standardized Waterfall

To be as fast and generic as possible, a standardized waterfall is built into the Accelerator and is designed to cover most of the customer needs. This waterfall looks like this:

Standard WF Structure

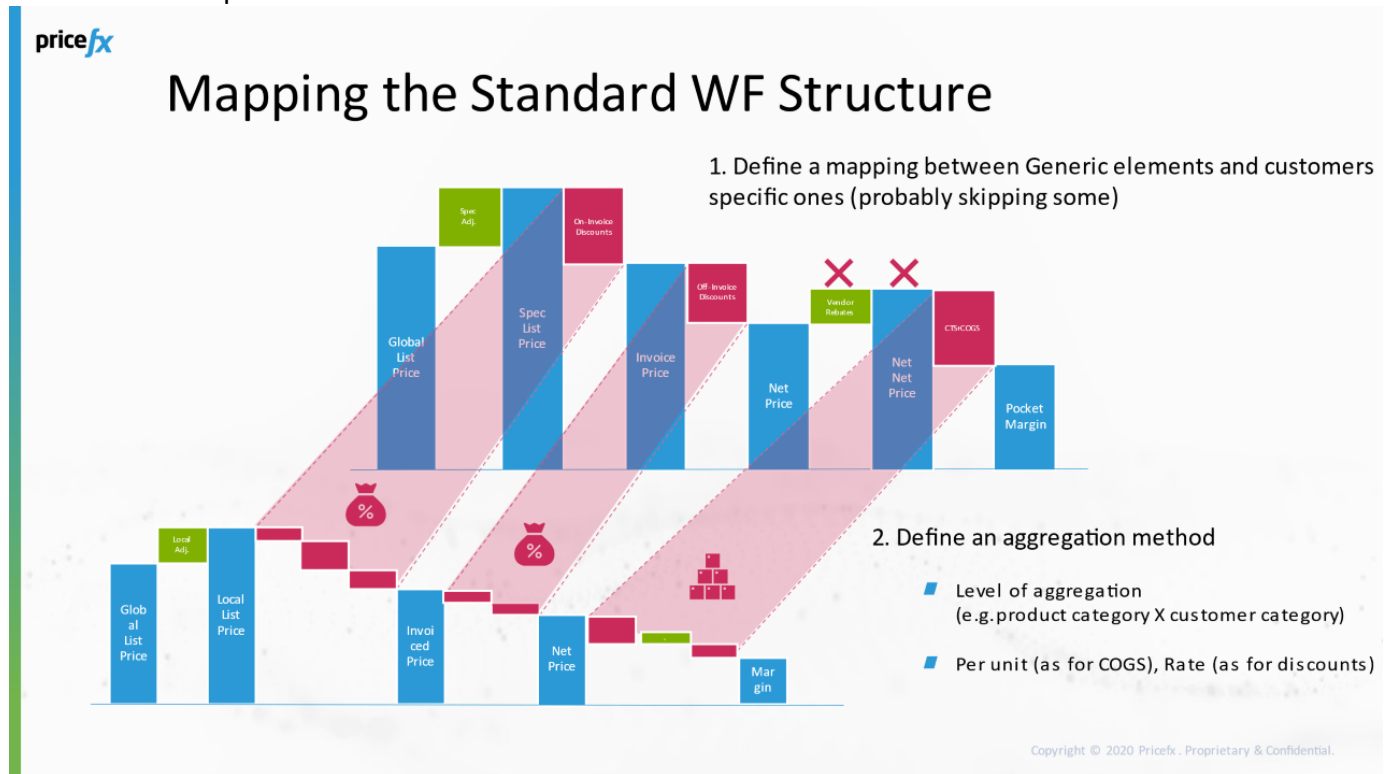


Built down to the level of Product X Customers

Copyright © 2020 Pricefx . Proprietary & Confidential.

A key part of the setup is mapping the actual customer waterfall columns to this standard one. This could mean that some of the elements of the customer waterfall will be aggregated within a single element of the standard waterfall and that some elements may not be used at all.

Illustrative example:



Copyright © 2020 Pricefx . Proprietary & Confidential.

As a result:

- Customers need to be aware of the fact that the **results will not reflect their exact waterfall**.
- We can standardize most things - mainly the way we consume results (Optimization Realization Dashboards, Price Setting, Agreements & Promotions, etc.).

What to Map and How

The prerequisite to the Accelerator deployment is the availability of a Datamart containing the transactions data and the fields listed below. You may need to create such a Datamart if the existing data are not directly available or integrate information from several distinct sources (e.g. Master Data + Data Source + Price Extension).

Transactions Datamart Prerequisites

- The transaction source must contain the following fields (create them if needed):
 - **Customer** field - ID of each customer. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it.
 - **Customer Group** field - ID of each customer group. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it. It must be defined as a dimension.
 - It must be a segmentation level of the segmentation model you will use in the optimization.
 - **Product** field - ID of each product. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it.
 - **Product Group** field - ID of each product group. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it. Must be defined as a dimension.
 - It must be a segmentation level of the segmentation model you will use in the optimization.
 - **Quantity** field - Number of products in each transaction.
 - **Global List Price** field - Global list price in each transaction (extended to quantity, POAI does not expect unit prices).
 - **Revenue** field - Also named Invoice Price in the POAI Accelerator waterfall (extended to quantity).
 - **Margin** field - Also named Pocket Margin in the POAI Accelerator waterfall (extended to quantity).
 - **Segmentation Target** field - See [Elasticity Model for PO.AI Accelerator](#).
- The transaction source can also contain the following optional fields for the POAI Accelerator waterfall. As the previous price in the waterfall can be used during the mapping in case the specific adjustment or off invoice discount or vendor rebate is null:
 - **Specific list price** field - List price after applying a specific adjustment to the global list price in each transaction (extended to quantity, POAI does not expect unit prices).
 - **Net price** field - Price after applying an off invoice discount to the invoice price in each transaction (extended to quantity, POAI does not expect unit prices).
 - **Net net price** field - Price after applying a vendor rebates to the net price in each transaction (extended to quantity, POAI does not expect unit prices).
- **Avoid null values** in dimensions because the corresponding transaction rows will be ignored by the elasticity model and will cause an error in POAI models. If necessary, create a new field by using an expression that replaces empty values with a text (e.g., "Not provided").

Mapping during the Accelerator Deployment

The mapping step during the Accelerator deployment in PlatformManager allows you to inform which columns from the transaction source have to be used to compute the aggregated data.

All numerical values are extended to quantity.

Settings

Source Type

Source Name

Product

Product Label

Product Group

Product Group Label

Customer

Customer Label

Customer Group

Customer Group Label

Quantity

Global List Price

Specific List Price

Invoice Price

Net Price

Net Net Price

Pocket Margin

[Continue](#)

[Cancel](#)

Options description:

- **Source Type** – Type of the transaction source. Always prefer to use a Datamart.
- **Source Name** – Name of the transaction source.
- **Product** – Column from a source containing the product ID.

- **Product Label** – Column from the transaction source containing the product name. If not provided, the product ID will also be used as the product name.
- **Product Group** – Column from the transaction source containing the ID of the group to which the product belongs.
- **Product Group Label** – Column from the transaction source containing the name of the group to which the product belongs. If not provided, the ID of the group will also be used as the product group name.
- **Customer** – Column from the transaction source containing the customer ID.
- **Customer Label** – Column from the transaction source containing the customer name. If not provided, the customer ID will also be used as the customer name.
- **Customer Group** – Column from the transaction source containing the ID of the group to which the customer belongs.
- **Customer Group Label** – Column from the transaction source containing the name of the group to which the customer belongs. If not provided, the ID of the group will also be used as the customer group name.
- **Quantity** – Column from the transaction source containing the number of products in the transaction.
- **Global List Price** – Column from the transaction source containing the global list price of the transaction (extended to quantity). The unit list price of each product will be inferred from this column.
- **Specific List Price** – Column from the transaction source containing the specific list price of the transaction (extended to quantity).
- **Invoice Price** – Column from the transaction source containing the invoice price of the transaction. The revenue will correspond to this column (extended to quantity).
- **Net Price** – Column from the transaction source containing the net price of the transaction (extended to quantity).
- **Net Net Price** – Column from the transaction source containing the net net price of the transaction (extended to quantity).
- **Pocket Margin** – Column from the transaction source containing the pocket margin of the transaction (extended to quantity).
The master unit cost of each product will be inferred from Pocket Margin, Net Net Price, and Quantity.

Dealing with Missing Fields

In some cases, you will want to instantiate a waterfall without all of the discounts. However, the fields are still required by the Accelerator to do its mapping, meaning that it will expect values you cannot give from a Data Source as they are not available in your problem. In this case, you can simply provide the same fields several times to simulate two sequential values separated by a discount of 0.

Example:

Quantity
Quantity

Global List Price
Global List Price

Specific List Price
Customer List Price

Invoice Price
Invoice Price

Net Price
Invoice Price

Net Net Price
Realized Price

Pocket Margin
Gross Margin

Continue Cancel

In this example, the column of the Datamart "Invoice Price" is used twice to compensate a missing "Net Price". The "Net net price" exists and the discount between it and the "Net Price" will be computed with the "Invoice Price" instead. The discount between the column "Invoice Price" and "Net Price" will always be 0, as they refer to the same value.

Elasticity Model for POAI Accelerator

To retrieve the elasticity factor calculated from transaction data, the POAI model deployed by the Accelerator depends on another segmentation model. Below is the simplest way to configure such a model on the partition; the complete documentation for segmentation models is in [How to Configure Segmentation-Optimization Models](#).

In this section:

- [Deploy](#)
- [Prepare Transaction Source](#)
- [Create Model](#)
- [Define Model](#)
- [Perform Analysis](#)
- [Calculate Segmentation](#)

Deploy

⚠ Note that if the segmentation ModelType & Logics have already been deployed on the partition, deploying it again might override specific changes done by a pricing scientist. Also if the Segmentation Model Type is already present in Optimization > Model Types, you can directly use it without doing the deployment again.

For now, the deployment can be done with the IntelliJ Idea Studio plugin. The Model Type and all calculation logics from the [po_segmentation repository](#) have to be deployed to the partition.

Prepare Transaction Source

The segmentation model runs using a transaction source. The transaction source must meet the following prerequisites.

- The transaction source must contain the following fields (create them if needed):
 - **Customer** field - ID of each customer. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it.
 - **Customer Group** field - ID of each customer group. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it. It must be defined as a dimension.
 - **Product** field - ID of each product. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it.
 - **Product Group** field - ID of each product group. Be careful: it cannot be a source key. If it is, duplicate the field to be able to access it. It must be defined as a dimension.
 - **Quantity** field - Number of products in each transaction.
 - **Revenue** field - Also named Invoice Price in the POAI Accelerator waterfall (extended to quantity).
 - **Margin** field - Also named Pocket Margin in the POAI Accelerator waterfall (extended to quantity).
 - **Segmentation Target** field - Margin percentage field, containing the quotient of *Margin Measure* divided by *Revenue Measure*.
- The user will define some segmentation levels.
 - These fields must be defined as dimensions in the transaction source.
 - Avoid null values because the corresponding transaction rows will be ignored in the segmentation model and will cause an error in POAI models. If necessary, create a new field by using an expression that replaces empty values with a text (e.g., "Not provided").

Create Model

1. In **Optimization > Models**, use the **Add Model** button.
2. Set a **Name** and a **Label**.
3. Select **Segmentation** as **Model Type**.
4. Click the **Create** button.

Define Model

In the Definition step, define the model parameters:

- **Transaction source** is the dataset used to calculate the segments. Once provided, some fields based on it appear:
 - **Transaction Filter** - Allows you to filter the data. In case you are not sure of the integrity of the transaction source, you can set filters for:
 - Positive cost, revenue, and quantity
 - Margin percentage (or another optimization target) in a realistic range, e.g., between -10% and 100%
 - **Customer Field** - Field mapped to *Customer* during Accelerator deployment.
 - **Product Field** - Field mapped to *Product* during Accelerator deployment.
 - **Volume Measure** - Field mapped to *Quantity* during Accelerator deployment.
 - **Revenue Measure** - Field mapped to *Invoice Price* during Accelerator deployment.
 - **Margin Measure** - Field containing the difference between the fields mapped to *Net Net Price* and *Costs* during Accelerator deployment.
 - **Optimization Target** - Margin percentage field, containing the quotient of *Margin Measure* divided by *Revenue Measure*.

 Do not forget to save the model before continuing. The button is at the top right corner.

Perform Analysis

In the Analysis step, the DataPrep tab is not mandatory but it is useful to have as it provides an overview of the data. It aims to materialize the data with all the filters applied. It can also substantially reduce the processing time of the next steps. Just click the **Calculate** button at the top right corner.

The resulting table appears on the right of the **Definition** step. You can check some values and the total number of rows.

The next Analysis steps are not mandatory, they only serve a data analysis purpose.

Calculate Segmentation

In the Calculation step, these tabs have to be calculated, in this order:

1. **Segmentation** tab - Select *Manual Attribute Selection* as **Segmentation Method**. As **Segmentation Levels** select the columns mapped to Product Group and Customer Group during the Accelerator deployment, and order them in this specific order. Last, click the **Calculate** button at the top right corner.
2. **Scoring** tab - Just click the **Calculate** button at the top right corner.
3. **Pricing** tab - Just click the **Calculate** button at the top right corner.
4. **Elasticity** tab - Just click the **Calculate** button at the top right corner.
5. **Policies** tab - Just click the **Calculate** button at the top right corner. After calculation, the list of policy records is displayed. In column A, each one contains the elasticity factor to be used by the optimization model. The BreakEven tab calculation is optional, in the POAI Accelerator context.

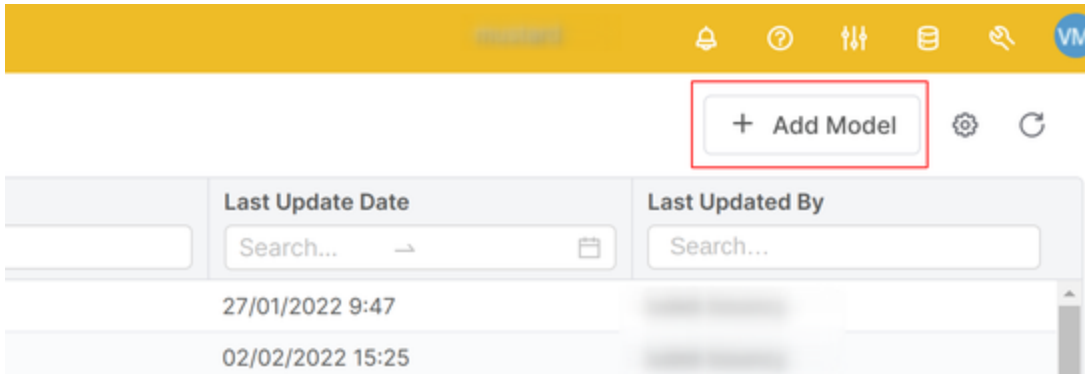
POAI Accelerator Run

To run the POAI Accelerator, you need to create a new Model Object from the **Optimization > Models (MO)** menu, using the POAI logic to instantiate the optimization model and give it the required parameters. The model will then be added to the list and will be editable.

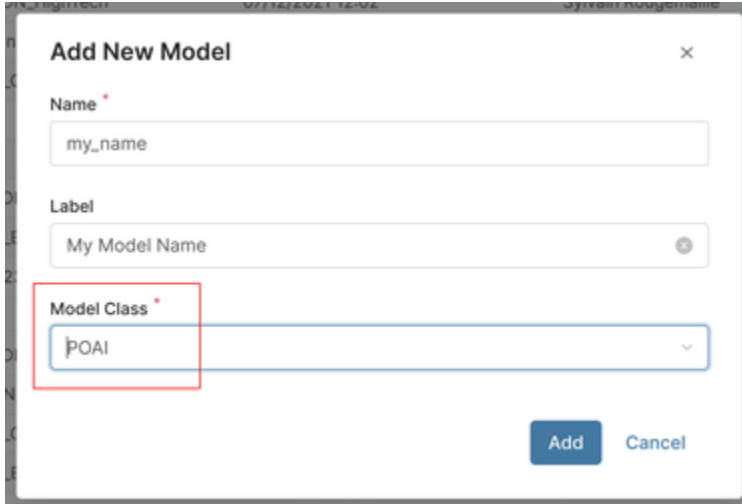
- [New Model](#)
- [Definition Step](#)
- [Scope Step](#)
- [Configuration Step](#)
 - [Boundaries Tab](#)
 - [Objectives Tab](#)
 - [Advanced Parameters Tab](#)
- [Results Step](#)

New Model

1. Go to **Optimization > Models (MO)** and add a new model.

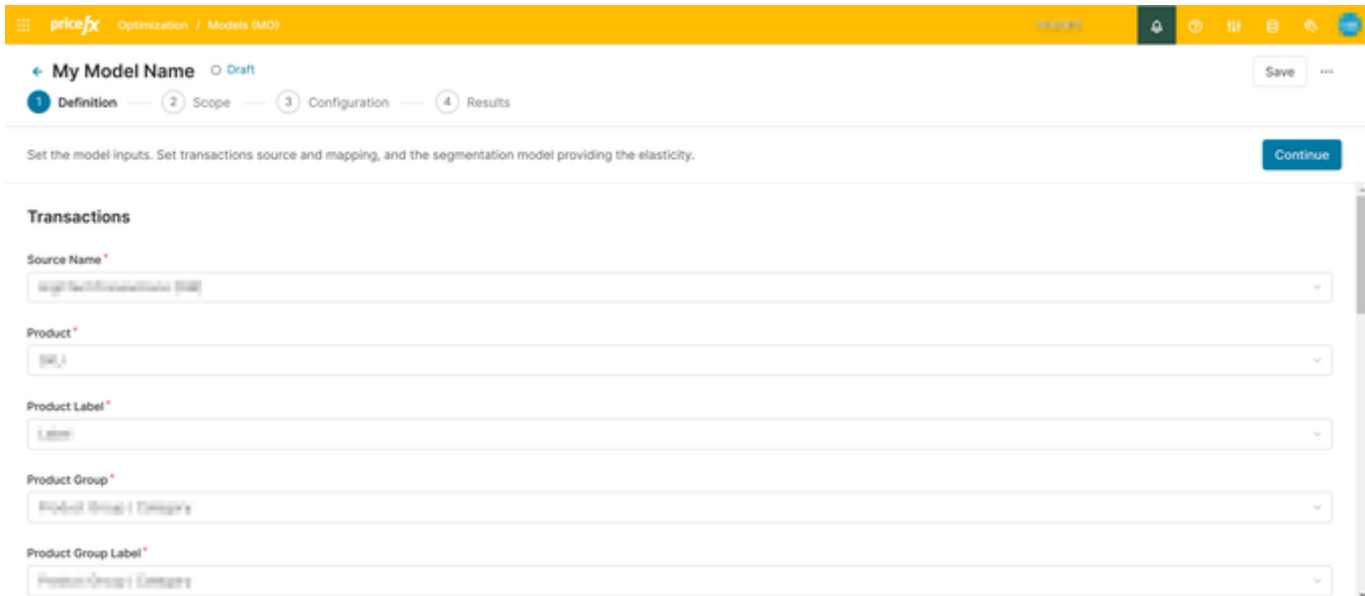


Define the name of the model and choose *POAI* as **Model Class**.



3. The new model opens. It is based on the data you defined when deploying the Accelerator. It automatically reads the data and wrangles them. Wait a moment, until the calculation finishes. Then you can interact with the model.

The interface is:



Definition, Scope, Configuration, and Results are the steps of the model and will be explained below. When you are in your model in the partition, you are always guided by the short description provided just below the name of the step.

Definition Step

This is the first step of the model. When you open it for the first time, the user inputs of the first section, called *Transactions*, are prefilled with the default values. It is the mapping of the data. You can change it if needed. The second section is called *Elasticity* and you define there the segmentation model to use in the optimization (see [Elasticity Model for POAI Accelerator](#)).

⚠ Choose a segmentation model among those on the partition *whose levels match* the Customer Group and Product Group fields you mapped during the deployment.

When the Definition step is done, you can go to the Scope step. For this, use the Continue button at the top right.

pricefx Optimization / Models (MO)

My Model Name Draft Save ...

1 Definition — 2 Scope — 3 Configuration — 4 Results

Set the model inputs. Set transactions source and mapping, and the segmentation model providing the elasticity. Continue

Revenue Price

Net Price *

Net Net Price *

Pocket Margin *

Elasticity

Segmentation Model *

If all required fields are not provided, there will be an error message.

pricefx Optimization / Models (MO)

My Model Name Draft Save ...

1 Definition — 2 Scope — 3 Configuration — 4 Results

Set the model inputs. Set transactions source and mapping, and the segmentation model providing the elasticity. Continue

Revenue Price

Net Price *

Net Net Price *

Pocket Margin *

Elasticity

Segmentation Model *

Cannot calculate next step because inputs of Mapping tab are not valid. It seems that some required inputs might be empty.

The Continue button will automatically run the calculation that starts the next step. After the calculation, which can take a few minutes, you can access the Scope tabs.

Scope Step

The calculation that runs the first time you go to the Scope step, or if you have changed the values in the Definition step, is mainly the aggregation of the transactions data by product and by customer, in order to perform at this level.

At the end of the calculation, the tab is available and there is a table called Wrangled Data in the model tables (accessible through the menu in the top right corner).

In the Scope step, you define which products and customers the optimization will apply to. This step works on already wrangled data, i.e. all the historical transactions have been aggregated by product and customer.

- **Included Customer Groups** and **Included Product Groups**: they rely on the customer group field and on the product group field which was defined during the POAI Accelerator deployment. If nothing is provided, there is no filter.
- **Customer Minimum Revenue**, **Customer Minimum Margin (%)**, **Product Minimum Revenue**, and **Product Minimum Margin (%)** are filters related to the global values.
- You can add as many custom filters as you want with **Data - Advanced Filters**.
⚠ Remember that these filters apply to the already wrangled data (at a customer x product level), only on the fields that exist in the wrangled data.

In the right panel, there is an overview of the data which will be taken into account in the optimization. It is refreshed to reflect your filters when you click the **Apply Settings** button.

Configuration Step

The Continue button at the Scope step will automatically run the calculation that starts the Configuration step. After a calculation, which can take a few minutes, you can access the configuration tabs. The calculation creates some model tables: Segmentation contains the elasticity values which will be used for each segment, and the Problem tables define the coordinates of the data for the optimization engine at different levels of granularity.

In the Configuration step, you define your optimization configuration.

- **Boundaries** tab defines how much you accept a change on the key variables of the model.
- **Objectives** tab defines the business objectives of the optimization (see below).
- **Advanced Parameters** tab defines the parameters of the optimization run itself.

Boundaries Tab

These entries define how much you allow the model to change each row value. Specific Adjustments, On Invoice Discounts, and Off Invoice Discounts boundaries are absolute values; while Invoice Prices boundaries are defined by a relative change from the historical values.

Objectives Tab

- **Revenue/Margin Optimization** takes a value between 0 and 1. The objective of the model is to maximize a mix between global revenue and global margin (profit). If the input is 0, it will maximize the global revenue; if the input is 1, it will maximize the profit; intermediate values allow to mix and weigh these two goals.
- **Revenue** objective is to define a revenue target for selected customer groups. You need to add a customer group to the input matrix. Then the current revenue of the customer group is displayed. You can enter a variation percentage to define the target revenue of the customer group; this value is

displayed in the Target Revenue column. You can also define different priorities (low, medium, high) for each customer group's revenue target objective. Only the customer groups in the input matrix have a revenue target objective. If you do not create any row in this input matrix, there is no revenue target objective in the model.

- **Volume** objective is to define a volume target for selected product groups. You need to add a product group to the input matrix. Then the current volume (sold quantity) of the product group is displayed. You can enter a variation percentage to define the target volume of the product group; this value is displayed in the Target Volume column. You can also define different priorities (low, medium, high) for each product group volume target objective. Only the product groups in the input matrix have a volume target objective. If you do not create any row in this input matrix, the model has no volume target objective.

Advanced Parameters Tab

The default values are 500 optimization steps, no maximum run time, and an early stop if the model is stabilized. If the model is not stabilized after 500 steps, you can increase the number of steps. If the model takes too long to run, you can add a maximum run time.

The Profiling checkbox can help you understand how the agents reach their optimum but it increases the need for memory and the optimization run time.

Results Step

When the Configuration step is done, you can go to the Result step, using the Continue button. It will first run the longest calculation of the model. This calculation is a sequence that you can follow in the job tracker:

pricefx Optimization / Models (MO)

My Model Name Draft

1 Definition — 2 Scope — 3 Configuration — 4 Results

Recalculate Save ...

Optimization results. Compare the current and optimized state at a high level and in detail, see how much business objectives and rules have been met.

Processing 3/4

Data needs to be calculated to show proper information in this step. This might take several minutes.

Calculation run: to track the jobs click the blue text (arrow)

First, the model runs some preparations. Then, two optimization engines are launched, one named Simulation, which defines the initial state of the optimization, and one named Optimization which performs the real optimization. In the end, postprocessing is run. The duration of the run depends mainly on the size of the input data in the scope and on the number of optimization steps.

Once the calculation has run, some other tables are available through the table link, but normally the user does not need to access them. If needed, go to the core optimization engine to understand what these tables are. The most important ones are the *Current* and *Optimized* tables that reflect the state of all the values before and after the optimization and have the same structure, and *Glassbox* tables that are used to dig into the way the optimization engine reached the optimized state.

The Results step tabs are:

- **Impact** tab - Displays comparisons between the current state (before optimization) and the optimized one.
- **Details** tab - Displays tables that compare the current to the optimized values at different levels of granularity.
- **Glassbox** tab - Displays charts that provide the state of the values finders and the criteria at the end of the optimization. It is useful to understand how the model reached its optimized state.
- **Query** tab - Displays the relevant optimized values for user inputs.

For more details see [POAI Accelerator Results](#).

POAI Accelerator Technical Content

This section details the ModelClass and the logics that the POAI Accelerator deploys. For each step, its aim, its outputs, and the main reasons to modify the logics are explained. If there is a need to modify the logics, refer to the process in [POAI Accelerator Customization](#) and to documentation in [Problem Modeling](#), [Problem Description](#), and [Problem Tables](#).

In this section:

- [POAI ModelClass](#)
- [Libraries](#)
- [Definition Step](#)
 - [Calculation: Wrangling](#)
 - [Scope Tab](#)
 - [Segmentation Source Tab](#)
- [Configuration Step](#)
 - [Calculation: Filtering](#)
 - [Boundaries and Objectives Tabs](#)
 - [Advanced Parameters Tab](#)
- [Results Step](#)
 - [Calculation: RunOptimization and PrepareResults](#)
 - [Impact Tab](#)
 - [Details Tab](#)
 - [Glassbox Tab](#)
 - [Query Tab](#)

POAI ModelClass

POAI ModelClass organizes a list of logics to create the model architecture. It is transformed in an optimized UI in the Pricefx platform. The general architecture is:

There are two types of logics: *calculation*, which writes tables in the model, and *evaluation*, whose purpose is only to display some results. The standard Model Class definition is documented in [Model Class \(MC\)](#).

All the logics of the POAI Accelerator follow a standard naming convention: first *POAI_* prefix, then the first letters of the step name, then *Calc* or *Eval*, depending on the formula nature, then the name of the tab. The evaluation logics that take the user input need also a corresponding configurator; it is defined in a logic of the same name, suffixed with *Configurator*. In the end, there are two library logics named *POAI_Library* and *POAI_InternalLibrary*.

Libraries

The logics are **POAI_Library** and **POAI_InternalLibrary**.

◦ Aim of the logics

POAI_Library is used in nearly all the other logics deployed by the Accelerator and defines a set of generic OE-related functions to facilitate model's tables reading and writing and to deal with parameters definition in the UI. It is accessed via the calls on `libs.POAI_Library.XXX` in the code. *It is preferable to keep this code unmodified* to facilitate the maintenance and readability of the project.

POAI_InternalLibrary contains some functions needed specifically for this Accelerator, such as reading its configuration from the application settings, applying the user filters in each part of the model, and many small helpers for the charts rendering.

◦ Common reasons to modify the logics

If a specific function is needed for the project, it should be added in `POAI_InternalLibrary`. It will be accessed then via `libs.POAI_InternalLibrary.XXX` in the code.

Definition Step

The calculation logic is `POAI_Def_Calc_Wrangling` and there are two tabs with related evaluation logics: **Source** and **Scope**.

Calculation: Wrangling

The logic is `POAI_Def_Calc_Wrangling`.

▼ Aim of the logic

This logic wrangles the transaction Datamart, which was defined before the Accelerator deployment. The optimization is *done at a product x customer level*, not at the transaction level.

▼ Outputs of the calculation

A Datamart table called Wrangled Data is created in the model. It is produced the customer transactions Datamart. This is the main connection to external data.

▼ Common reasons to modify the logic

This calculation is the main connection to the external data and most often requires to be modified to accommodate the specifics of the customer data.

The wrangled data are the result of an aggregation. In some cases, the aggregation can change; for example to replace an average with a median.

If the waterfall structure is not the standard one, maybe some columns should be added to the wrangled data.

Scope Tab

The logic are `POAI_Def_Eval_Scope` and `POAI_Def_Eval_Scope_Configurator`.

▼ Aim of the logics

These logics, whose content is mainly included in the configurator, let the end user choose the filters to scope the optimization. It means, for example, creating a model with a scope filtered on a product group or a set of customers.

▼ Outputs of the evaluation

A map of filters which are called later in the code by `model.inputs('definition', 'scope').Scope`.

A dashboard with information, charts, and tables that summarize the scope taken into account according to the user filters.

▼ Common reasons to modify the logics

The main reason to modify these logics is to retrieve data from other sources (PX, DS, DM...) or with different filters. The filtering options that we expose to the user are modified here.

Segmentation Source Tab

The logics are `POAI_Def_Eval_Source` and `POAI_Def_Eval_Source_Configurator`.

▼ Aim of the logics

These logics, whose content is mainly included in the configurator, let the end user choose the segmentation model. The segmentation is used to retrieve the elasticities. In the code, we set the dimensions which are required in the segmentation to solve the problem to filter out the irrelevant ones.

▾ Outputs of the evaluation

The segmentation model name. It is called later in the code by: `model.inputs('definition', 'source').Source.SegmentationModel`.

▾ Common reasons to modify the logics

This tab and related logics can be removed if no segmentation-related outputs are needed.

Configuration Step

Calculation: Filtering

The logic is **POAI_Conf_Calc_FilterScope**.

▾ Aim of the logic

The goal of this calculation is to transform the wrangled data into a collection of tables stored in the model and to present the data efficiently for the Optimization Engine. It involves mainly SQL requests, used with Pricefx API functions, as well as shortcut functions to filter the data and read the Accelerator settings from the library `POAI_InternalLibrary`. This step also extracts the elasticities from the Segmentation Model in a table.

▾ Outputs of the calculation

This step writes a collection of tables; the most important ones are:

- Segmentation table - Produced from the selected Segmentation model.
- `Problem_nameOfTheSpace_nameOfTheScope` tables - Data manipulation to prepare the last tables needed by the Optimization Engine. These logics are prefixed by "Store_" and create the model tables that act as an endpoint for the Optimization Engine. Be careful, *their names follow a strict format*. Only the problem tables that do not contain user input values, like objective and constraints, are created here. The other ones are created during the `RunOptimization` calculation.

▾ Common reasons to modify the logic

The complexity of the data preparation needed in this step depends on the problem to solve. The main reason to modify this logic is to expose different Problem tables to the optimization engine if some spaces or some scopes are changed in the problem.

Boundaries and Objectives Tabs

The logics are **POAI_Conf_Eval_Boundaries_Configurator** and **POAI_Conf_Eval_Objectives_Configurator**.

▾ Aim of the logics

The Boundary and the Objectives tabs are used to retrieve the user objectives and as such, are configurators. They contain all the information needed to guide the optimization process by setting the constraints and the goals to reach. The separation into two tabs is mainly for UI purposes.

▾ Outputs of the evaluation

The user inputs are stored to be aggregated in the following steps with the rest of the data. In general, it means computing some target and threshold values at different levels of granularity.

▾ Common reasons to modify the logics

It is quite common to change these logics by adding or modifying the constraints and objectives in the problem; for example, adding targets at some levels or setting thresholds to keep some values in

check. These modifications are *needed but not sufficient* as the problem modeling itself must be changed to take them into account. It is possible to change the number of the tabs where these objectives and constraints are defined, but then the Model Class definition has to be modified too.

Advanced Parameters Tab

The logic is `POAI_Conf_Eval_AdvancedParameters_Configurator`.

◦ Aim of the logics

This tab is used to retrieve the optimization run parameters and as such, is a configurator.

◦ Outputs of the evaluation

The user inputs are stored to be used in the `RunOptimization` calculation. The most important values are the following three that define when the optimization engine will stop:

- **Max. Number Of Steps** - For how long, in a number of steps, the optimization will run before considering it has reached a "good" state. The default value should be enough, but remember that this value can be increased if the optimization logs many values as "still converging".
- **Max. Number of Minutes** - For how long, in a number of minutes, the optimization will run before considering it has reached a "good" state. The default value is 0 (= no limit), but for some large models it may be useful to define a default limit;
- **Stop when stabilized** - Allows an early stop if the model is stabilized.

There is also a Profiling group of parameters but it is more for a development purpose.

◦ Common reasons to modify the logics

The most common reason to change the logic is to set different default values. All the outputs of this logic are used in the `POAI_Res_Calc_RunOptimization` logic.

Results Step

Calculation: `RunOptimization` and `PrepareResults`

The logics are `POAI_Res_Calc_RunOptimization` and `POAI_Res_Calc_PrepareResults` and they are run in this order.

RunOptimization Calculation

◦ Aim of the logic

The goal of this calculation is to create a simulation and an optimization run and retrieve their results. To do so, we need to create a [Problem Description](#) that details the structure of the problem to solve by the Optimization Engine and to give endpoints for the OE to get the data of the problem. The previous steps will change the problem by altering its scope and changing the objectives, and the data will be fed directly to the OE thanks to the model tables.

This step consists of:

- **Data manipulation** to prepare the last tables needed by the OE. These logics are prefixed by *Store_* and create the model tables prefixed by *Problem_* that act as an endpoint for the OE. Be careful, *their names follow a strict format*: These endpoints must be named according to the `Problem_nameOfTheSpace_nameOfTheScope` present in the `ProblemDescription.groovy` and return the corresponding data. The behavior of the OE and its way of reading data from endpoints highlight the need for a well-thought-out Scope step. Creating tables of the needed data, already computed and aggregated, implies being well aware of "where is the data I need" and "how do I need to transform it". Due to computations depending on user inputs from business rules and objectives,

some tables have to be created in later steps, explaining why even the RunOptimization calculation can have some *store* logics. That is why it is normal to refactor and improve scope logics and the other store logics during the development of *ProblemDescription.groovy*.

- **ProblemDescription.groovy** element - Returns the problem description in a map. The content of the problem description is detailed in [Problem Description](#).
- **ProblemData.groovy** element - Parses the problem description to automate the post-processing.
- **Run.groovy** element - Contains the code that handles the problem description. It takes the description of the problem and the advanced parameters user inputs, and triggers the two optimization jobs thanks to `model.startJobTriggerCalculation`. Each run will return prefixed tables of similar structures. The jobs will run in parallel. The first one is the optimization itself and its outputs are prefixed by "Optimized". The second one is a simulation: it simulates the first state of the optimization and will be a reference to compare before/after values in the results dashboards. Its outputs are prefixed by "Current". The job type (optimization vs. simulation) is indicated by the input parameters of the `model.startJobTriggerCalculation` function.

Once the problem description is created, it is sent via a Kafka message to trigger the instantiation of a job running an OE configured by this file. The OE has to have access to the correct endpoints to get the data and to know where to write back the results when the computation is finished.

▾ [Outputs of the calculation](#)

The Groovy code does some preparation work. It creates `Problem_nameOfTheSpace_nameOfTheScope` tables - data manipulation to prepare the last tables needed by the OE. These logics are prefixed by "Store_" and create the model tables that act as an endpoint for the OE. Be careful, *their names follow a strict format*. Only the problem tables that contain user input values, such as objectives and constraints, are created here. The other ones are created during the `POAI_Conf_Calc_FilterScope` calculation.

A Groovy element also reads the problem description to retrieve a list of parameters used during the postprocessing step to reformat the Glassbox data.

At the end of its run, the OE will write a set of model tables containing its results and the Glassbox information needed to understand why this solution was used. This writing is done directly by the OE and is *not related to a Groovy logic*. This job is done by the two OE jobs, the simulation one and the optimization one.

- The Glassbox table provides optimization indicators for each pair of instantiated value finder - criterion. The simulation job does not create any Glassbox table.
- The tables prefixed by *Results_* present the state of the objectives and constraints at the end of the optimization.
- The tables prefixed by *Solution_* present the raw values that the system was *meant to find* (declared as `Value_Finder` in the Problem Description). The simulation job does not create any Solution table.
- The tables prefixed by *Simulation_* present the value of computed variables marked as exposed in the description, typically including *values of interest* such as forecasted quantities.

▾ [Common reasons to modify the logic](#)

Any modification of the problem modeling and type of constraints to apply or objectives to reach implies a modification of the Problem Description, thus *ProblemDescription.groovy* should change accordingly.

⚠ If the problem description changes, do not forget to check if it is necessary to change or create some Problem tables, either in the Configuration step, `POAI_Conf_Calc_FilterScope` logic, or here in the Results step, `POAI_Res_Calc_RunOptimization` logic.

In some cases, it could be useful to change the OE image and/or the OE tag that the job trigger refers to. Their values are in the element *Run.groovy*.

PrepareResults Calculation

▼ Aim of the logic

This calculation retrieves the outputs of the RunOptimization logic and reformats them to provide tables that can be used to show user-friendly optimization results. Each element stores one model table or some similar tables.

- Store_Optimized and Store_Current logics create the Optimized and Current tables that mirror each other. The Current table details the state of the system before the optimization; the Optimized one is the state after the optimization. Therefore a lot of visualizations will directly use these two tables to highlight the evolution of key values and the impact of the optimization.
- Store_Details_ logics write tables whose goal is to clearly show, for each type of variable adjusted by the OE, the optimized value, the reference value (what would that value be if the OE did not optimize anything), their difference, and the needed data to know what is impacted by this value (for example, returning the SpecificAdjustmentRate, but also the revenue and the margin rate, by customer group in the Customer Groups table). This way, the user can see the impact of the optimization on every adjusted value.
- Store_Glassbox_ logics calculate aggregated metrics on the optimization agents criteria and value finders.

▼ Outputs of the calculation

This calculation writes a collection of tables:

- Current table - Details the state of the system before the optimization.
- Optimized table - Details the state of the system after the optimization. Its architecture mirrors the Current table one.
- Details tables - There is one table for each type of variable adjusted by the OE, and it contains the optimized value, the reference value (what would that value be if the OE did not optimize anything), their difference, and the needed data to know what is impacted by this value (for example, returning the customer_id and the customer_type for the Detail_CustomerAdjustment table). This way, the user can see the impact of the optimization on every adjusted value.
- GlassboxVF_ tables - Their names are built as GlassboxVF_NameOfTheSpace_NameOfTheValueFinder, there is one table by value finder key (i.e. type of value finder). These tables store the overall values of each value finder.
- GlassboxProbe_ tables - Their names are built as GlassboxProbe_NameOfTheSpace_NameOfTheValueFinder, there is one table by value finder key (i.e. type of value finder). These tables store the initial movement of each value finder.
- GlassboxCriteria_ tables - Their names are built as GlassboxCriteria_NameOfTheSpace_NameOfTheCriterion, there is one table by criterion key (i.e. type of criterion). These tables store the overall values of each criterion.
- Glassbox_AggregatedMetrics table - Summarizes the global interaction indicators between each value finder key and each criterion key.
- Glassbox_VFs_by_Key table - Summarizes the global overall indicators of each value finder key.
- Glassbox_Criteria_by_Key table - Summarizes the global overall indicators of each criterion key.
- Glassbox_Spaces table - Summarizes the total number of criteria and value finders in each space.

▼ Common reasons to modify the logics

If the problem description has been changed, then the parameter set that helps to create the Current and the Optimized tables has to be changed too. It is in the element *AggregationUtils.groovy*.

The other most common reason to change the logic is to reformat some data to ease the work of providing charts in the Result step tabs.

Impact Tab

The logic is **POAI_Res_Eval_Impact**.

▼ Aim of the logic

This tab exposes the results of the OE execution in an HTML summary and some complex graphs. It is a comparison of the optimized values and the current ones (what would that value be if the OE did not optimize anything).

The charts are created with the usual method:

```
api.newChartBuilder(...)
```

▼ Outputs of the evaluation

There are eight portlets:

- HTML summary to write explicitly the main variations.
- Waterfall comparison.
- Four bar charts that compare the revenue and margin, before and after the optimization, by product group and by customer group.
- Two bullet charts, one for the current state and one for the optimized one. They show the volume, the revenue, and the margin percentage for each pair of product x customer.

For details see <https://pricefx.atlassian.net/wiki/spaces/ACC/pages/4188438873/POAI+Accelerator+Results#Impact-Tab>.

▼ Common reasons to modify the logic

Add, modify or remove visualizations. This step is one of the most straightforward ones and its modification should not impact the previous steps.

Details Tab

The logic is **POAI_Res_Eval_Details**.

▼ Aim of the logic

This tab clearly shows, for each type of variable adjusted by the OE, the optimized value, the current value (what would that value be if the OE did not optimize anything), their difference, and the needed data to know what is impacted by this value. This way, the user can see the impact of the optimization on every adjusted value.

▼ Outputs of the evaluation

Some data tables. For details see <https://pricefx.atlassian.net/wiki/spaces/ACC/pages/4188438873/POAI+Accelerator+Results#Details-Tab>.

▼ Common reasons to modify the logic

Add, modify or remove table outputs. In practice, if the list of the value finders or their definition in the problem description change, the detail tables would change and thus these outputs would be modified.

Glassbox Tab

The logic is **POAI_Res_Eval_Glassbox**.

▼ Aim of the logic

This tab exposes the technical state of the OE execution at the end of the process. It is a development tool to help tune the model.

▼ **Outputs of the evaluation**

This logic displays charts that show the satisfaction, influences, impacts of the value finders and the criteria, initial movements of the value finders, and evolution of the criticality during the process of optimization. For details see <https://pricefx.atlassian.net/wiki/spaces/ACC/pages/4188438873/POAI+Accelerator+Results#Glassbox-Tab>.

▼ **Common reasons to modify the logic**

In general, there is no reason to change this dashboard.

Query Tab

The logic is **POAI_Res_Eval_Query**.

▼ **Aim of the logic**

This logic is used to access model results from outside of the model itself; for example in another logic. The first step is to use `api.model("ModelName")` to get the model and then use the function `evaluate` on it to retrieve an answer depending on the nature of the given parameters. The code needed to get these results is:

```
def model = api.model("The Model Unique Name")
def results = model.evaluate(
  "query_results",
  [
    product: "someProductID",
    customer: "someCustomerID",
  ]
)
def unitNetPrice = results['unit_net_price']
```

The accepted inputs and the corresponding returned results are:

Inputs	Results
product	<ul style="list-style-type: none"> unit global list price product group
customer	<ul style="list-style-type: none"> on invoice discounts rate customer group and all from customer_group
customer_group	<ul style="list-style-type: none"> specific adjustments rate
product and customer_group	<ul style="list-style-type: none"> unit-specific list price all from product all from customer_group
product and customer	<ul style="list-style-type: none"> unit invoice price off invoice discounts rate unit net price

- vendor rebates rate
- unit net net price
- unit costs
- unit pocket margin
- and all from above

▼ Outputs of the evaluation

The evaluator provides all the optimized waterfall values. For details see <https://pricefx.atlassian.net/wiki/spaces/ACC/pages/4188438873/POAI+Accelerator+Results#Query-Tab>.

▼ Common reasons to modify the logic

If the optimization model depends on new fields and if it provides new values, the evaluator should be changed to take them into account.

POAI Accelerator Results

Once a model has been run, the Results step contains four tabs:

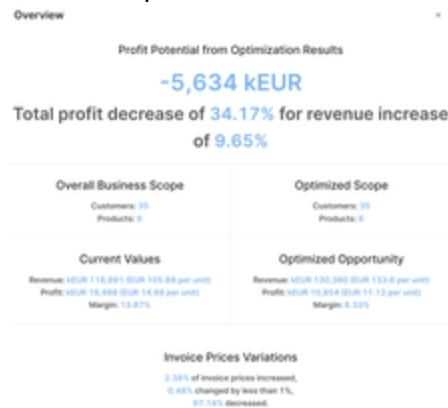
- [Impact Tab](#)
- [Details Tab](#)
- [Glassbox Tab](#)
- [Query Tab](#)

Impact Tab

This tab displays comparisons between projections with the current pricing and projections with the optimized one. So, be careful, the “current” charts represent the initial state of the Optimization Engine and not historical values from the past. They show what the Optimization Engine thinks would happen if the historical prices and discounts would be kept as they are. There are eight portlets in the dashboard, described below.

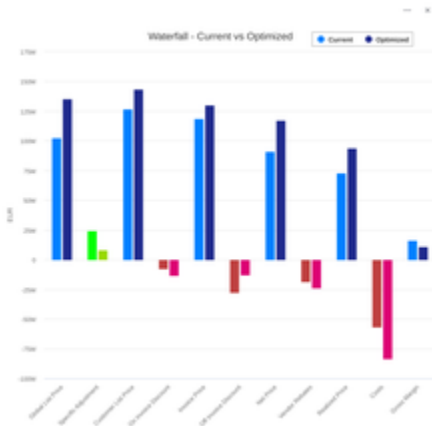
Overview

The overview provides a summary of the difference between the projections from current pricing and the projections from optimized pricing: global revenue and profit, but also the count of increased, stable, and decreased prices.



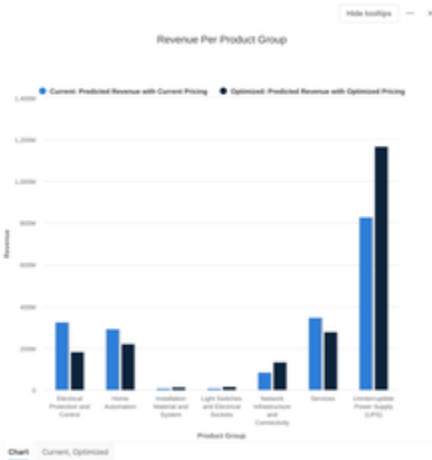
Waterfall

The waterfall displays both current and optimized waterfall (in parallel). All the values are extended ones, within the scope of the optimization. They are presented in a bar chart, hence negative values like discounts are below the X axis.



Revenue and Margin by Product and Customer Groups

These four portlets display the extended values of revenue and profit, aggregated by product and customer groups, to allow the end user to see where the optimization had more or less impact on the results.



Pocket Margin % vs. Volume

Two portlets display bubble charts, one with the current values and one with the optimized ones. Each bubble is a pair (product, customer). The size of the bubble is the revenue, the position of the bubble is the sold quantity on X, and the margin % on Y axis. A regression line shows the tendency. The color of the bubble of a given pair is the same in both portlets.



Details Tab

The Details tab displays the results tables in an easy way for the user to interact with. Different aggregations are provided: by product, by customer, by product and customer, and by customer group. Each table provides values for all the fields that are calculated at this level of granularity. The values provided are the historical, current, and optimized ones, plus the delta between the current and the optimized value. If needed, you can export these tables to Excel.

For the record:

- *historical* means the aggregated value coming from the source Datamart;
- *current* means the initial state of the Optimization Engine;
- *optimized* means the final state of the Optimization Engine.

Customer Group

Table

Customer Group	Historical Revenue	Current Revenue	Optimized Revenue	Revenue Δ	Historical Profit
Industry	933,075,551	939,937,760	1,349,691,369	409,753,609	183,100,330
Distributor	759,190,954	790,459,591	977,579,832	187,120,241	127,041,576
Direct Customer	203,161,470	208,930,860	443,044,950	234,114,089	45,967,899

Glassbox Tab

This tab's target audience is Configuration Engineers, Business Analysts, and team members working on improving a new optimization model. The tab provides insights to understand how the Optimization Engine reached its final state. One needs to understand the main concepts of the Optimization Engine to benefit from this dashboard. Two interesting pages to help users are [OE Glossary](#) and [Explainability \(Glassbox\)](#).

There are eleven portlets in this dashboard:

- two about [criteria satisfaction](#) (Satisfaction and Satisfaction by Criteria Type),
- two bar charts about the [interactions between value finders and criteria](#) (Impact vs. Satisfaction and Impact vs. Influence),
- two [bubble charts](#) about the value finders' influences and the criteria impacts,
- two [boxplot charts](#) about the value finders' variations and two others about the initial movements of the value finders,

- and optionally a chart of the evolution of the criteria agents during the optimization process.

The data for some of these charts are grouped by “agent key”. An agent key depicts the nature of the agent within the waterfall. For instance, `UnitGlobalListPrice` is a key. In the Accelerator, the agent representing the unit global list price for any specific product is a value finder, so we call it a “value finder key”. Criteria also have keys, like `RevenueTarget` for instance.

Criteria Satisfaction

The whole goal of the Optimization Engine is to satisfy criteria. These charts enable the user to assess at a glance whether or not the engine was successful at it.

The Satisfaction pie chart displays the total number of criteria by satisfaction status (satisfied, acceptable, unacceptable). The meaning of this status is explained in [Criteria Description](#).



This count itself is not sufficient because some criteria may be much more important than others. For instance, the global `RevenueMarginMixMaximization` criterion is only a single criterion but it impacts the whole scope of the optimization, whereas there is the `MaxUnitInvoicePrice` criterion for each of product x customer pair but each one is individually of lesser importance. This is why a bar chart provides more details, by displaying the satisfaction status by criteria key and space, i.e. by criterion type and its level of granularity. The tooltips also indicate how many criteria are instantiated.



Interaction Between Value Finders and Criteria

Value finders and criteria are both ends of a feedback loop, their interactions are fundamental to the convergence of the system towards a solution. These charts provide insights about these interactions.

The first bar chart displays data per criteria keys x value finders keys. Each criterion has an impact on some value finders and is more or less satisfied at the end of the run. The satisfaction is not related to any specific value finders, it is only related to the criteria. The *Impact vs. Satisfaction* bar chart is sorted by total impact amount. The more a criterion is impactful on a value finder, the higher it is in the chart. The expected output is to have more satisfied criteria at the bottom of the chart: these criteria do not impact the value finders because they are satisfied.

Questions one may want to ask:

- Why does a criterion have not much impact if it is not totally satisfied? It could be because the criterion has an impact mostly on another value finder key. Does it make sense for this customer or this model?
- Why does a criterion have much impact on a value finder that is already mostly satisfied or acceptable? It may be because of its importance or because of interactions.



While the criteria have an impact on the value finders, the value finders also influence the criteria. The second bar chart, which is also at a granularity level of criteria key x value finders key, compares the criteria's impact (on value finders) to the value finders' influence (on criteria). The Y axis indicates the pairs *criteria keys - value finders key*. It is also sorted by impact in descending order, hence it is the same order as the previous chart. You can check that high impact is generally correlated with high influence, although that may be scrambled by the various priorities and levels of granularities handled by the model.

Questions one may want to ask:

- Which criteria have more impact on which values finders?
- Which value finders have more influence on which criteria?

It can help tune the priorities of the criteria.

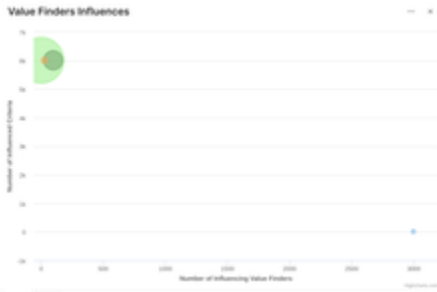


Bubble Charts

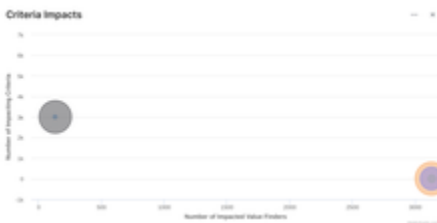
Two bubble charts are displayed, offering a quick view of the scale of value finders and criteria, both in raw quantity and in terms of impact/influence. They both have the same axis but they do not use the same data. The X axis represents the number of value finders, and the Y axis the number of criteria. The first bubble chart is aggregated by the value finders keys, the second one by the criteria keys.

The *Value Finders Influences* bubble chart displays a bubble for each value finder's key, whose size represents the overall influence of the value finders with this key on the criteria of the model. The coordinates represent how many value finders provide this influence and how many criteria they act on.

For instance, in the example above, few value finders have much influence on many criteria and many value finders (blue bubble) have not much influence on few criteria.



The *Criteria Impacts* bubble chart displays a bubble for each criteria key, whose size represents the overall impact of the criteria with this key on the value finders of the model. The coordinates represent how many value finders are impacted by this criterion and how many criteria act.



Boxplot charts

The boxplot charts are represented by the value finder key. A single box is a rectangle starting from the first quartile and ending in the third one, with a horizontal line representing the median. The whiskers have a maximum length of 1.5 times the difference between the median and the quartile it starts from. Moreover, each point of data is represented by a gray dot, slightly jittered horizontally. Hence, if some dots are beyond the whiskers, it means that the data has outliers. For all the boxplot charts, some useful information:

- There is a legend on the right of each chart, that separates boxplots from the scatter plot. Some value finder keys represent many points of data. Clicking on the legend "Scatter" will remove the dots and let the user see the box better.
- There is a tooltip for each box on mouse-over. It provides the minimum value, the three quartiles, including the median, and the maximum value. The minimum and maximum values may not be the extremities of the whiskers (if there are outliers).
- Sometimes the boxes have very different heights, for instance when a value finder key represents some prices or even revenues, and another one represents some percentages like discounts or margins. It is always possible to zoom in on the charts. The user has to simply grab a rectangular with their mouse.

The first two boxplot charts represent the variations of the value finders. Each box represents a value finders' key and the scatter plot represents the density of the values. The tooltip provides the quartiles of the data, from minimum to maximum, and the space where the value finders' key is defined. The color code is shared by all the boxplot charts (but not with the previous bubble charts). I.e. a given value finder's key is represented by same color in all the boxplot charts but may be represented by another bubble color in the value finders bubble chart.



The *Value Finders Variations* boxplot chart shows the raw variation of the value finders. It tells the user which value finders move more and if there is a pattern in their variations.

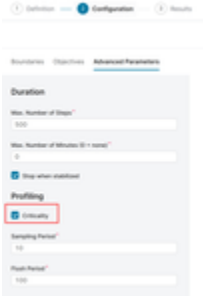
The *Value Finders Relative Variations* boxplot chart shows the same variations but in percentage. It is useful mainly to focus on small values, like the discount rates, that could have moved a lot relative to their initial value.

The *Initial Movements* boxplot chart represents the variation caused by the “probe” of the value finders, i. e. the initial action the value finder does on its own, without being pushed by any neighbor. This chart allows engineers to check if these probes are small enough not to disrupt the convergence.

The *Initial Movements Relative Weights* boxplot chart represents how prevalent the probe-induced movement is, compared to the overall variation of the value finder. Hence the users can assess whether the value finder has been mainly pushed by its criteria or disrupted by its probe.

Evolution of Criteria Satisfaction (Option)

As it involves a large amount of data, the last portlet is only created if the “Profiling” option in the Advanced Parameters (Configuration step) is checked.

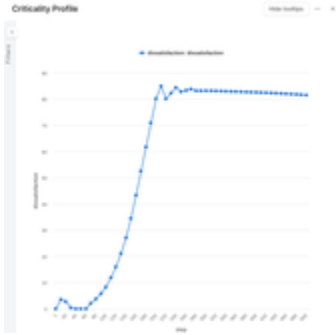


This chart displays the evolution of `max(dissatisfactions_in_the_sample)` during the run, but not all criteria are present in each sample. Hence, each value may correspond to a different criterion. This curve lets the user know whether the convergence is finished or it still needs some more steps. It is also a good tool to tune the various value finders parameters (for details see [Parameters](#)).

It is not a really good tool if maximization criteria are present in the problem. These criteria work in a special way and often have a negative dissatisfaction, making the profile curve hard to read. For instance, in the image below, the chart alternates between a highly dissatisfied criterion (in the 500M-1000M range), a moderately dissatisfied one (near zero) and a maximization one with negative dissatisfaction.



The user can filter this chart, for instance, by groups of criteria. In the chart below, we can see the evolution of the criterion which corresponds to the “near-zero values” on the overall scope chart above. We see that this criterion’s dissatisfaction increased (probably the system was helping other criteria at that time), but it decreases by the end of the run and probably needs more steps to settle.



Query Tab

This tab simulates the evaluation logic that can be called from any other module. One portlet represents one visible element of the logic.