Accelerate Shelf Prices Optimization

Version 1.0

April 2023

Accelerate Shelf Prices Optimization

The Optimization - Shelf Price Accelerator provides a fast implementation of a B2C shelf prices optimization model. The use case is detailed in the Overview section and the other sections explain how to use the accelerator, depending on your role.

In this section:

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

Overview (Optimization - Shelf Price)

Purpose

Optimization - Shelf Price Accelerator intends to optimize the **shelf prices** for **end customer**, typically of a **B2C company**. This accelerator enables price optimization based on:

- Maximization of the revenue and/or profit
- Setting shelf prices limits
- Aligning prices across sales channels
- Adjusting sold quantity depending on price changes (thanks to an advanced elasticity model)
- Aligning rules with competitor prices
- Managing stock coverage expectations

Outputs

The output of the optimization model are shelf prices by product, store and time period. These shelf prices are displayed in the interface and can be fetched in any part of Pricefx solution and also queried by external systems through APIs.

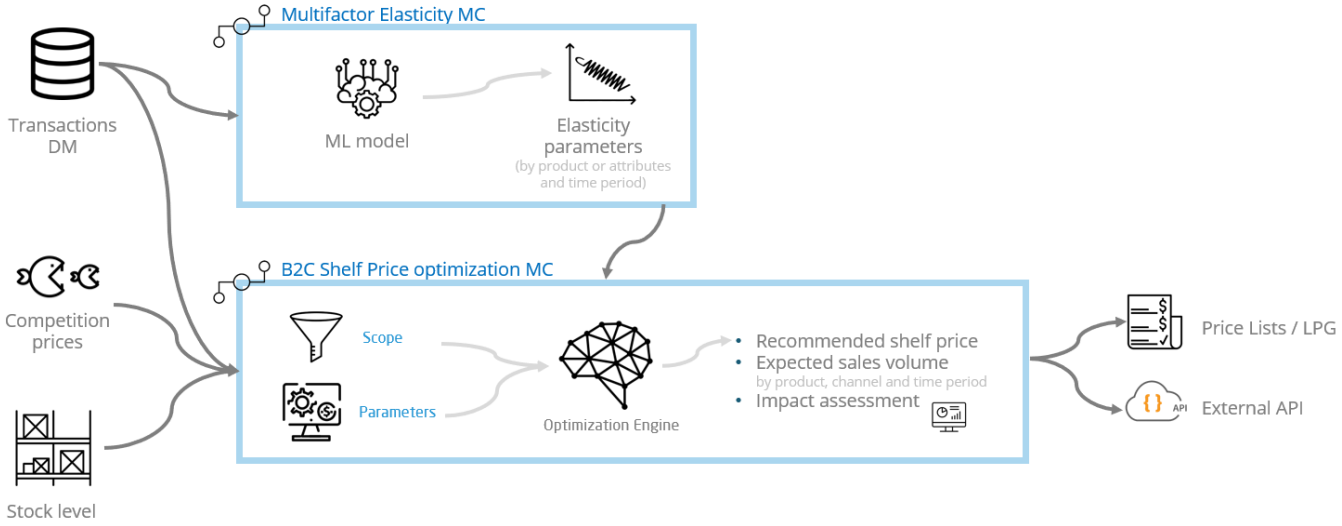
A set of dashboards is also provided to allow you to review and assess the outputs with charts and tables with the recommended shelf price, forecasted revenue and margin, stock coverage etc.

Approach

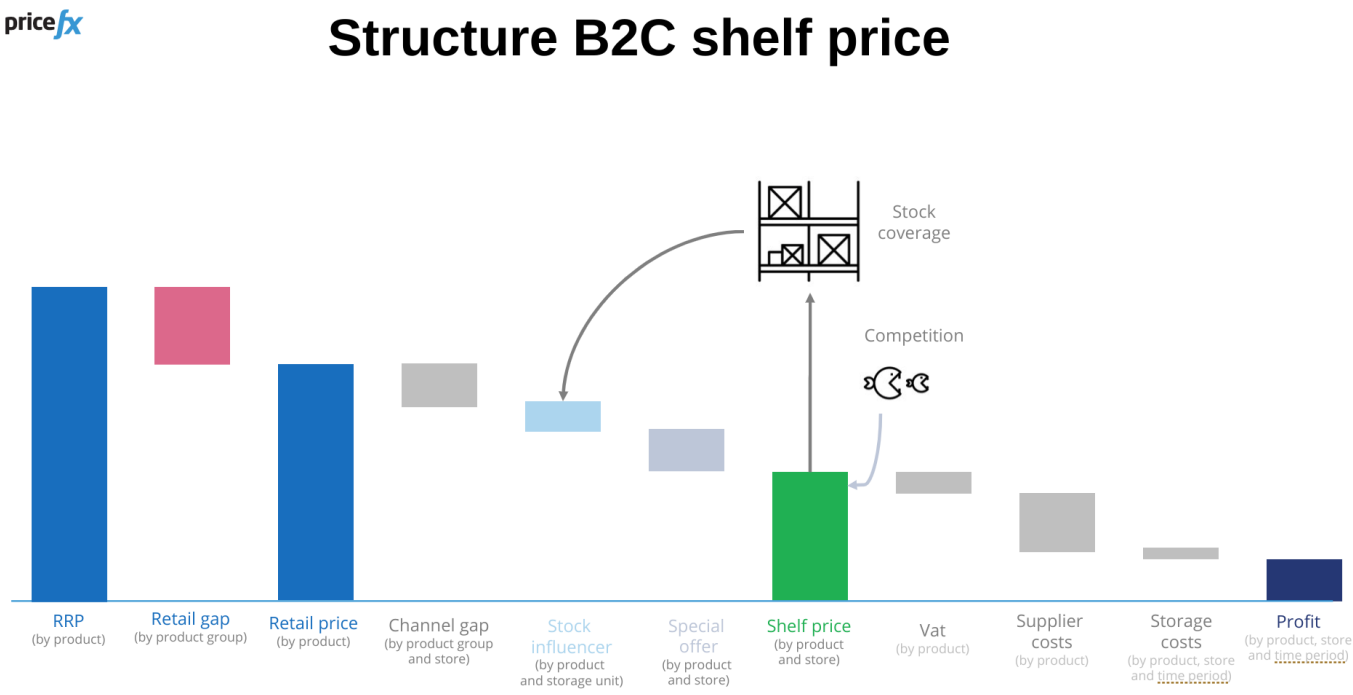
The optimization relies on two parts:

- Powerful elasticity model, built with Optimization - Multifactor Elasticity Accelerator, deployed at the same time as the Shelf Price Optimization
- Shelf Price Optimization itself

Here is an overview of the two models working together with inputs and outputs:



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:



With this data structure in place and the model parameters properly set, most of the work is done by the Optimization Engine, which positions the shelf price based on the price boundaries, competition prices, stock coverage etc. in order to optimize revenue and/or margin.

For details see Optimization Engine.

Limitations

- **Time period** – It is key to manage properly time periods, meaning selecting full time period of the transactions scope, both for the multi-factor elasticity and shelf price optimization. Partial period will result in incomplete results and wrong forecast due to biased inputs.
- **Product scope** – Relies on past transactions, so a product with no sales will not be part of the optimization.
- **No predefined extension point** – There is no out-of-the-box extension point defined for now. If you intend to add specific features, custom code should be written. (But then the accelerator becomes specific and it cannot be updated without extra effort to port those modifications.) Do not hesitate to report specific requirements and possible extension points to Pricefx.
- **Data requirements** – See Data Requirements (Clustering).

Business User Reference (Optimization - Shelf Price)

This section explains how to run a shelf price grid optimization and how to understand its results. Before you can run the Optimization - Shelf Price Accelerator, you must have it deployed on the partition. See Installation (Optimization - Shelf Price) for details. A model for the elasticities must be run before the optimization (covered in the Accelerate Multifactor Elasticity Optimization section).

- Data Requirements (Optimization - Shelf Price)
- Prepare Elasticity Model (Optimization - Shelf Price)
- Usage (Optimization - Shelf Price)
- Result Description (Optimization - Shelf Price)

Data Requirements (Optimization - Shelf Price)

These are the data prerequisites to run a Shelf Price model.

Three sources are needed, they can be either Datamarts or Data Sources. They can have mixed types.

- **Sales** Datamart or Data Source – Represents transactions, special events, and product attributes.
- **Stock** Datamart or Data Source – Contains the historical stock quantity by store.
- **Competition** Datamart or Data Source – Contains the list of competitors prices.

Sales

Field	Required?	Comment
Product	Yes	
Product Name	No	
Store	Yes	If there is only one store, set always the same value.
Store Name	No	
Channel	Yes	If there is only one channel, set always the same value.

Channel Name	No	
Period Start Date	Yes	The date of the start of week for the transaction.
Product Group	Yes	The product groups or categories.
Product Pareto	Yes	In general, the values are: 1 Best Seller, 2 Runner, 3 Core, 4 Long Tail... Using a prefix helps to keep them ordered as they are displayed alphabetically. But the user can have different category names.
Quantity	Yes	
Recommended Retail Price	Yes	Extended to the quantity.
Retail Gap	Yes	Extended to the quantity. If not existing for now, set to 0.
Retail Price	Yes	Extended to the quantity.
Channel Gap	Yes	Extended to the quantity. If not existing for now, set to 0.
Stock Influencer	Yes	Extended to the quantity. If not existing for now, set to 0.
Special Offer	Yes	Extended to the quantity. If not existing for now, set to 0.
VAT	Yes	Extended to the quantity. If you have the VAT rate, calculate the extended VAT from it.
Shelf Price	Yes	Extended to the quantity.
Revenue	Yes	Extended to the quantity. Without the VAT.
Supplier Costs	Yes	Extended to the quantity.
Storage Costs	Yes	Extended to the quantity. If not existing for now, set to 0.
Profit	Yes	Gross Margin extended to the quantity.
Minimum Advertise Price	No	Unit value

Stock

Field	Required?	Comment
Product	Yes	Must be of the same type as the Product field of the Sales source.
Store	Yes	Must be of the same type as the Product field of the Sales source.
Quantity in stock	Yes	Stock with a quantity of zero shall be included. Null values should be avoided.

Date of stock	Yes	At least one date of stock must exist for any product and store. The stock history is better to compute and display some metrics. The update frequency depends on the context, the best is the same frequency as the update frequency of the model.

Competition

Field	Required?	Comment
Product	Yes	Must be of the same type as the Product field of the Sales source. Only one competitor product should be defined by product and store.
Store	Yes	Must be of the same type as the Product field of the Sales source.
Competitor	Yes	
Shelf Price	Yes	Unit value, representing the competitor shelf price.
Date of competition price	Yes	It could be a "From Date", meaning from which date this price should be considered. The update frequency depends on the context, the best is the same frequency as the update frequency of the model.

Prepare Elasticity Model (Optimization - Shelf Price)

To retrieve the elasticities calculated from transaction data, the Shelf Price Grid model deployed by the Accelerator depends on another Multifactor Elasticity model. Below is the simplest way to configure such a model on the partition; the complete documentation for multifactor elasticity models is in Accelerate Multifactor Elasticity Optimization.

In this section:

Deploy

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

As the Optimization - Shelf Price Accelerator has a dependency on the Optimization - Multifactor Elasticity Accelerator, the quickest way to deploy both is to directly deploy the first one. It will take care of deploying the other if it has not yet been deployed on the partition. See Installation (Optimization - Shelf Price) for more details.

It is still possible to deploy only Optimization - Multifactor Elasticity; see Admin User Reference: Installation (Optimization - Multifactor Elasticity) for more details.

Prepare Transaction Source

The Multifactor Elasticity 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):
  - **Period Start Date** – Date value used to aggregate by period.
  - **Product** – ID of each product.
  - **Shelf Price** – Price applied to the customer (extended to the quantity).
  - **Quantity** – Number of products in each transaction.
  - **Store** – Store where the transaction was done.
  - All the fields that define the factors that influence the price elasticity of the products.

It is recommended that this transaction source is the one used in the Shelf Price Grid model to define the historical sales.

Create and Run Model

The documentation to create and run a Multifactor Elasticity model is in Business User Reference (Optimization - Multifactor Elasticity).

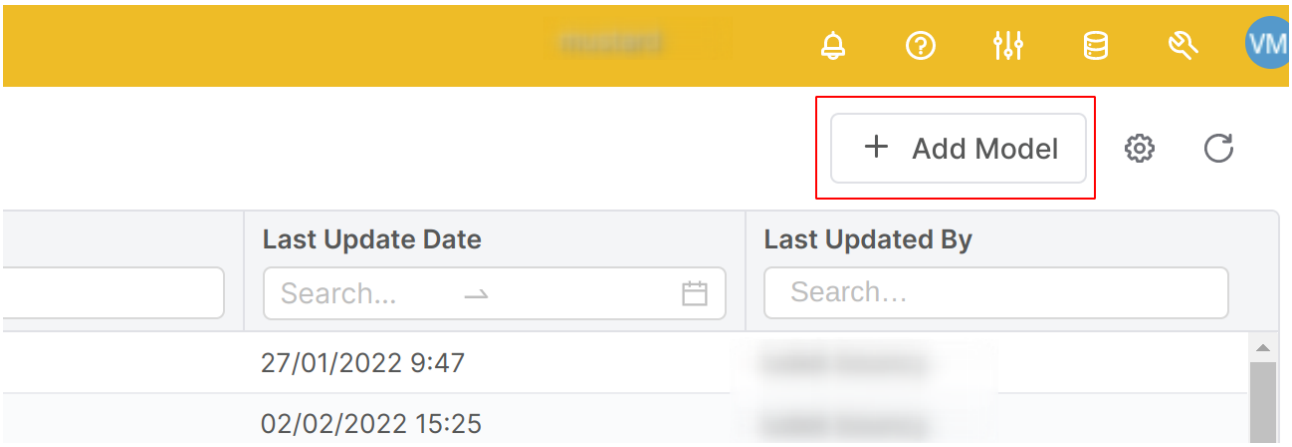
Usage (Optimization - Shelf Price)

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

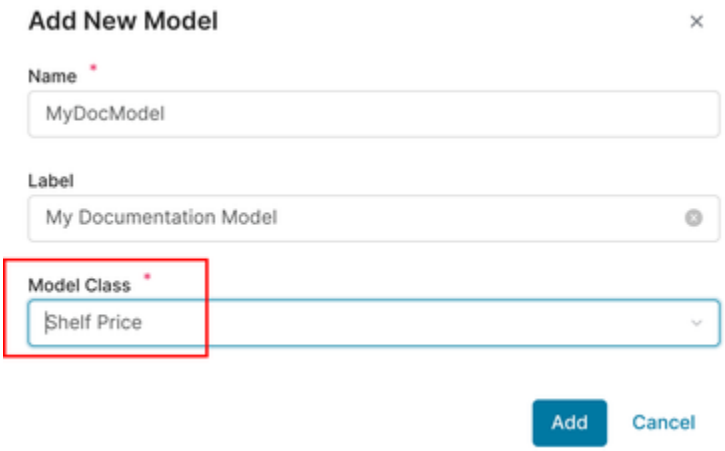
- New Model
- Definition Step
- Scope Step
- Configuration Step
- Results Step

New Model

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

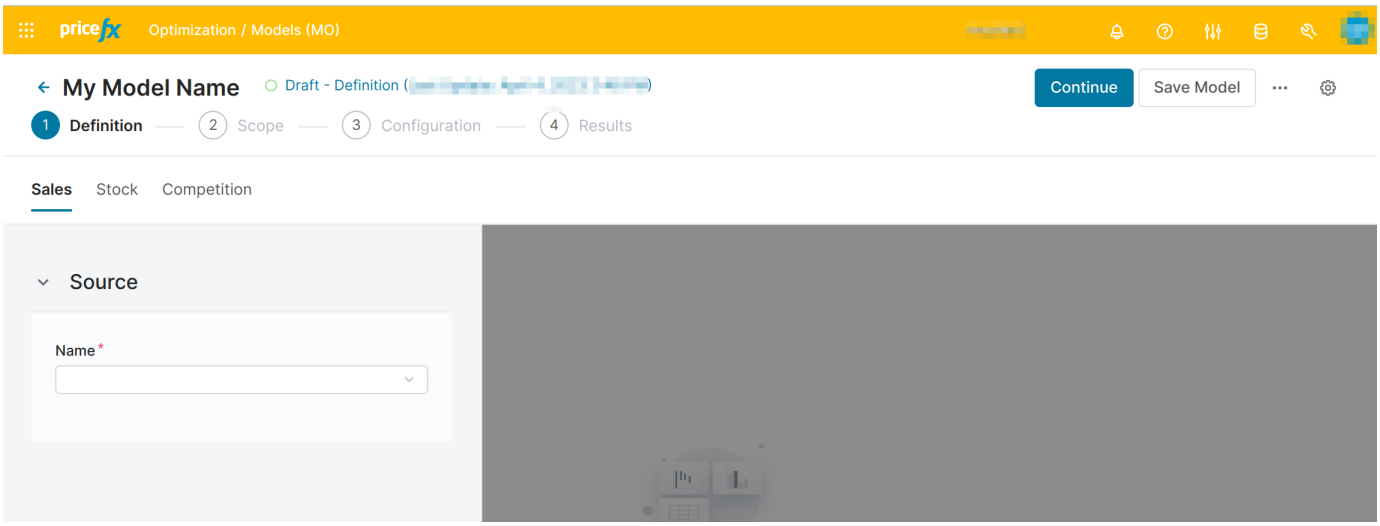


2. Define the name of the model and choose *Shelf Price* as **Model Class**.



3. The new model opens.

The interface is:



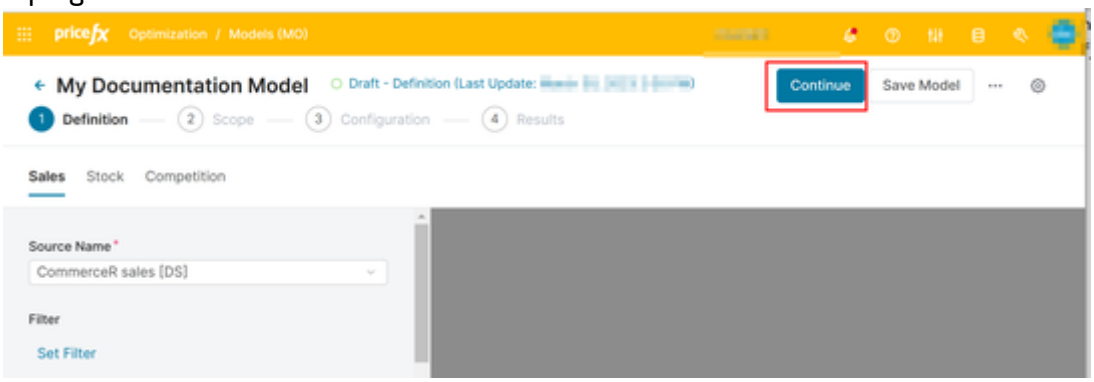
Definition, Scope, Configuration, and Results are the steps of the model and will be explained below.

Definition Step

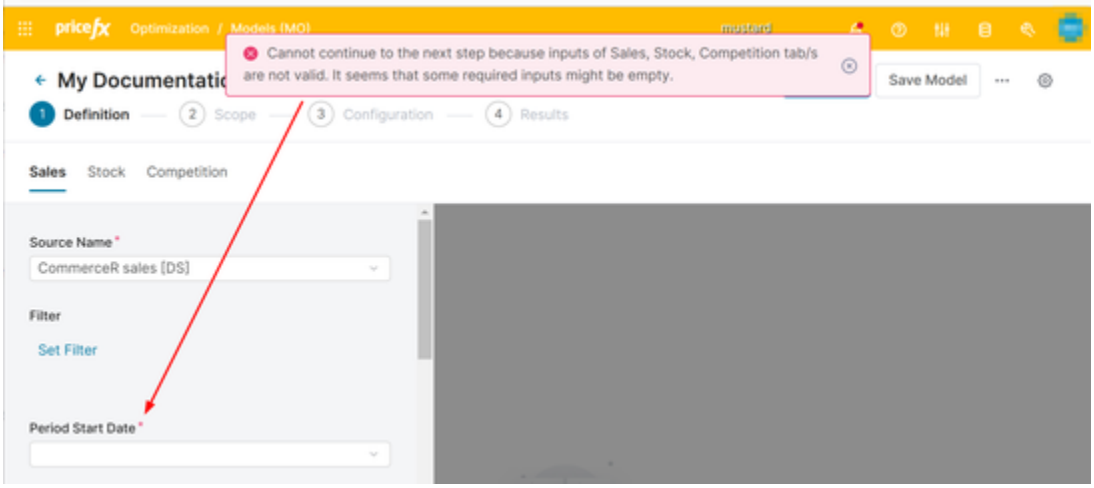
This step aims to define the input data and their mapping. You can also apply a filter to your data. There are three tabs:

- **Sales** defines the historical transactions.
- **Stock** defines the historical stocks.
- **Competition** defines the prices of the competitors on similar products.

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



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

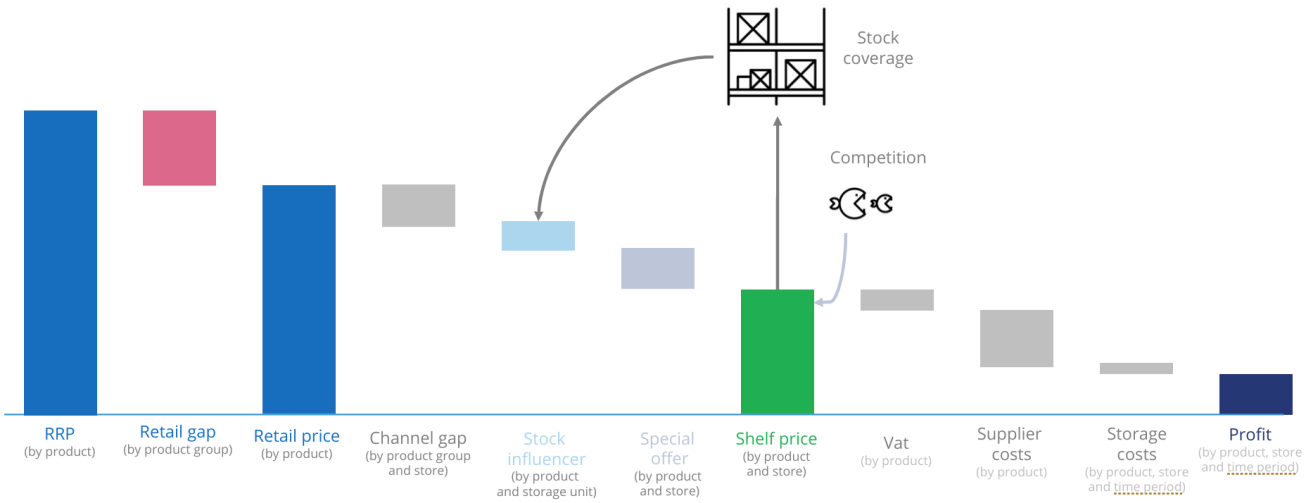


Sales Tab

In this tab, you define the transaction source and map it. The general recommendations are:

- The model automatically filters some unavailable values (like null, negative or zero values, depending on the fields). First, apply the configuration entries and check the filtered-out data before adding more filters, if needed.
- The dimension names (like *Store* or *Product*) are IDs, and the dimension names are the labels displayed in the interface. When there is no name field mapped, the ID is used instead.
- The *Period Start Date* is not the date of the transaction, but the date defining the period of the transaction, aggregated with the level of granularity needed – in general, it is by week.
- The money fields must provide extended values (except the Minimum Advertise Price which is not in the *Extended fields* section).
- The mapping of the fields should reflect this waterfall structure:

Structure B2C shelf price



Stock Tab

In this tab, you define the stock source and map it. The general recommendations are:

- The model automatically filters some unavailable values (like null, negative or zero values, depending on the fields). First, apply the configuration entries and check the filtered-out data before adding more filters, if needed.
- The *Product* and *Store* fields are used to link the stock data to the sales data. Their values and their types should correspond.

Competition Tab

In this tab, you define the competition source and map it. The general recommendations are:

- The model automatically filters some unavailable values (like null, negative or zero values, depending on the fields). First, apply the configuration entries and check the filtered-out data before adding more filters, if needed.
- The *Product* and *Store* fields are used to link the stock data to the sales data. Their values and their types should correspond.
- The *Shelf Price* field is the unit shelf price of the product sold by the competitor.

Scope Step

The Continue button at the Definition step will automatically run the calculation that starts the Scope step. This calculation mainly performs the materialization of the sales, stock, and competitor tables in the model. At the end of the calculation, the tab is available and there are three tables called *Sales, Stock*, and *Competition* in the model tables (accessible through the menu in the top right corner).

In the Scope step, you define the scope of the optimization.

- *Included Product Groups* and *Included Channels*: they rely on the product group field and on the channel field which was defined during the Definition step. If nothing is provided, there is no filter.
- *Product Minimum Historical Revenue, Product Historical Minimum Margin (%), Store Minimum Historical Revenue*, and *Store Historical Minimum Margin (%)* are filters related to global values.

- You can add as many custom filters as you want with *Advanced Filters*.

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.

Note that data shown in portlets are limited to scope, while filters are applied to historical values on the Sales table level. It might result in counter-intuitive behavior of the Product list and Store list portlets, hence the presence of historical and in scope columns.

Configuration Step

The Continue button at the Scope step takes you to the Configuration step. There is no calculation, so it is direct. In the Configuration step, you define your optimization configuration.

- The **General** tab defines the global inputs for the optimization
- The **Price** tab defines the limits that each shelf price should respect.
- The **Competition** tab defines the desired positioning with the different competitors.
- The **Stock** tab defines the desired coverage duration.
- The **Channels** tab defines the desired relative positioning between the different sales channels.

General Tab

The *Elasticity Model* is a model of type Multifactor Elasticity (see Accelerate Multifactor Elasticity Optimization), that must run until the end. The elasticity is used to forecast the quantity sold, for each product, store, and channel, when the price changes. This value is used to predict the stock coverage and to maximize the global revenue or profit.

Number of Optimized Periods defines how many forecasted weeks are calculated.

Optimization Priority defines the mix of global revenue vs. global profit that you want to maximize.

Price Tab

Shelf Price Changes Limits define how much the optimized shelf prices, by product, store, and period can differ from the historical values.

Shelf Price Limits define the minimum percentage value of the elements of the waterfall.

Enforce Minimum Advertise Price is to use the mapping field Minimum Advertise Price, in the Definition step, to enforce a minimum value to the shelf price of each product. It is not displayed if the field is let empty in Definition step.

Competition Tab

Acceptable Range From Competition Reference Price is a pair of values applied on each shelf price to position it with the competition price.

Competition Positioning Target is set for more global positioning. It defines how much the model should differ, on average, from each different competitor. The list of competitors is automatically created, based on the competition source.

Stock Tab

The stock coverage user inputs are used in combination with the elasticity model to set the prices that reach these stock coverage values. They are defined and applied globally for each product pareto category (e.g. 1 Best Seller, 2 Runner, 3 Core, 4 Long Tail) and period of time (week). The lines of the input matrix are automatically created to reflect the existing values for the Product Pareto field in the sales source and the default stock coverage value can be changed.

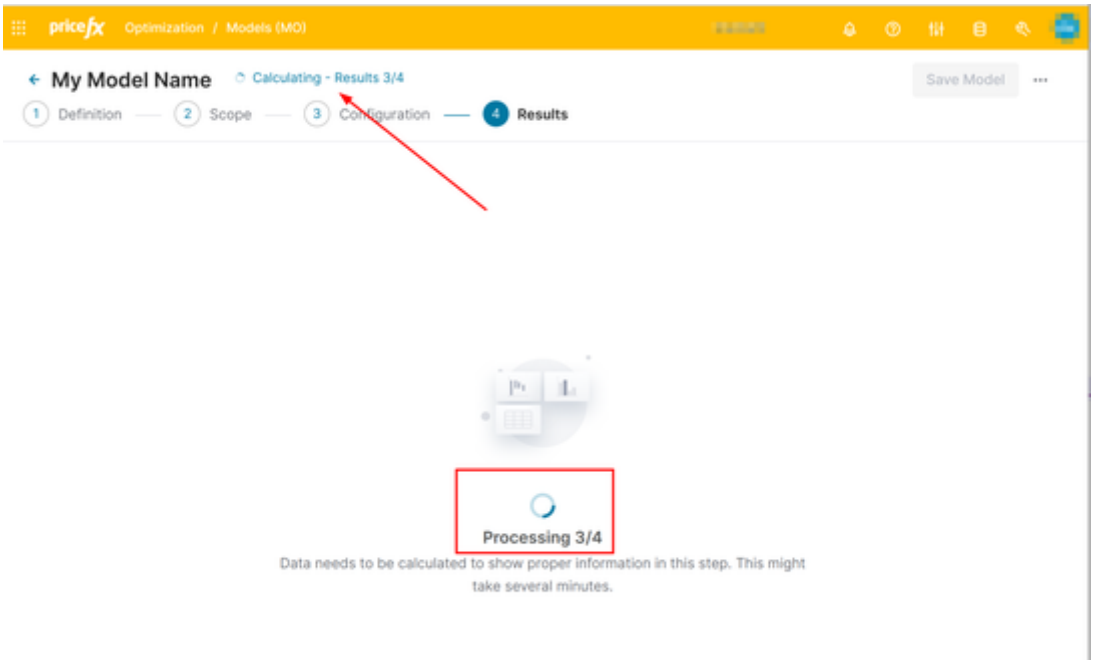
The stock coverage value (time before running out of stock) is always expressed in days.

Channels Tab

The Channels tab sets the alignment rules between channels. The user chooses a reference channel and sets the average gap percentage between the reference and any other channel. These gaps are used to define how much the average price by product of a channel should differ from the average price of the same product in the reference channel.

Results Step

When the Configuration step is done, you can go to the Results 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:



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 actual optimization. In the end, the postprocessing is run. The duration of the run depends mainly on the size of the input data in the scope.

Once the calculation has run, some other tables are available through the table link, but normally you don't need to access them. If needed, go to Optimization Results to understand what these tables are. The most important one is the *Optimized* table which reflects the state of all the values after the optimization, and the *Glassbox* tables which are used to dig into the way the Optimization Engine reached the optimized state.

The Results step tabs are:

- **Impact** - Displays comparisons between the current state (before optimization) and the optimized one.
- **Details** - Displays tables that compare the current to the optimized values at different levels of granularity.
- **Glassbox** - 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.
- **Evaluation** - Mocks the evaluation method of the model.

For more details see Result Description (Optimization - Shelf Price).

Result Description (Optimization - Shelf Price)

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

- Impact
- Details
- Glassbox
- Evaluation

Impact

This tab displays comparisons between the values of the last historical period and the first forecasted one. There are seven portlets in the dashboard, described below.

The scope of the impact dashboard can be set, using the user inputs on the left panel.

- **Overview** portlet provides a summary of changes in the shelf prices using the optimized values.
- **Self Price Change** portlet is a histogram showing how many prices increased or decreased.
- **Revenue Breakdown** portlet outlines what are the levers that explain the revenue change.
- **Competition Positioning** portlet displays the spread of the percentage difference of prices between the optimized shelf prices, by product and store for next period, and the competitor prices. It is useful to see how much the competition objectives set in the Configuration step could be reached by the Optimization Engine.
- **Channel Gaps** portlet displays the spread of the difference of channel gap between any channel and the reference one for any product and channel for next period. It is useful to see how much the channel objectives set in the Configuration step could be reached by the optimization engine.
- **Average Stock Coverage** portlet displays the evolution of the stock coverage, meaning the number of days before running out of stock. It is useful to see how much the stock coverage objectives set in the Configuration step could be reached by the optimization engine.
- **Revenue and Margin** portlet displays a historical chart, with a different color for the forecasted periods.

Details

The Details tab effortlessly displays the results tables for the user to interact with. Different aggregations are provided: global, by product, by product and store, and by product, store, and period. The user inputs allow filtering of all the data provided in the tables.

Each table provides values for all the fields that are interesting at this level of granularity. The values provided are the last historical period and the earliest forecasted one, plus the delta between the next and the previous period. If needed, you can export these tables to Excel.

For the record:

- *previous period* means the last historical period;
- *next period* means the first forecasted period.

The forecasted waterfall for the next period is also displayed in this tab.

Glassbox

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 three portlets about criteria satisfaction: Satisfaction, Satisfaction by Criteria Type, and Criteria Comparison. 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 data for these charts are grouped by "agent key". An agent key depicts the nature of the agent within the waterfall. For instance, `GrossMarginLowerThreshold` is key. All the criteria representing the margin lower threshold for any set of channel, date, product, and store refer to this shared key.

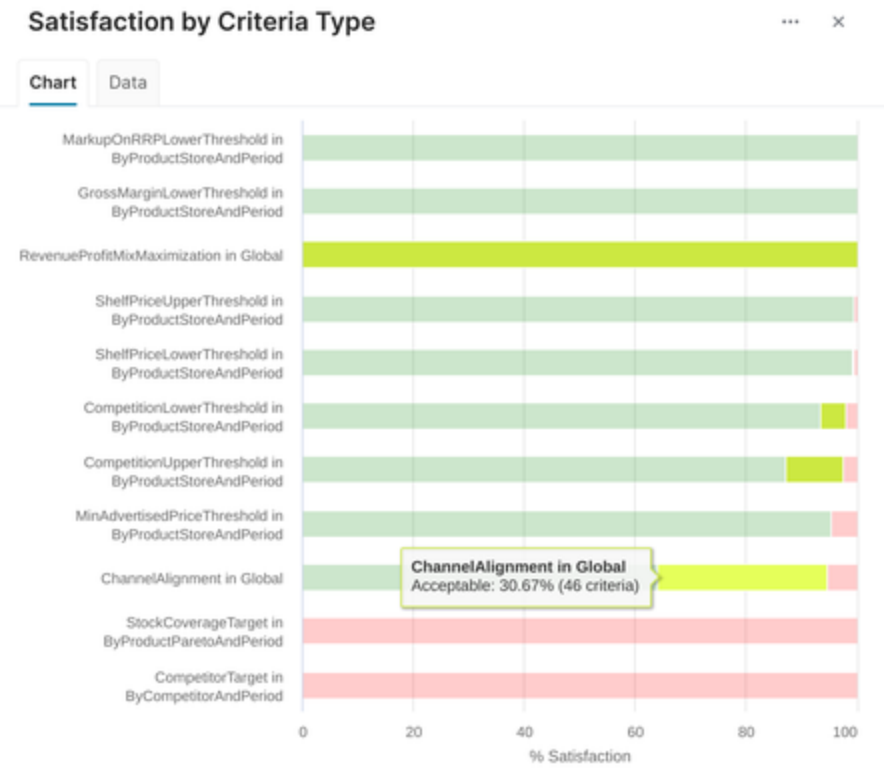
Satisfaction

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, the two next charts provide more clarity about the criteria satisfaction.

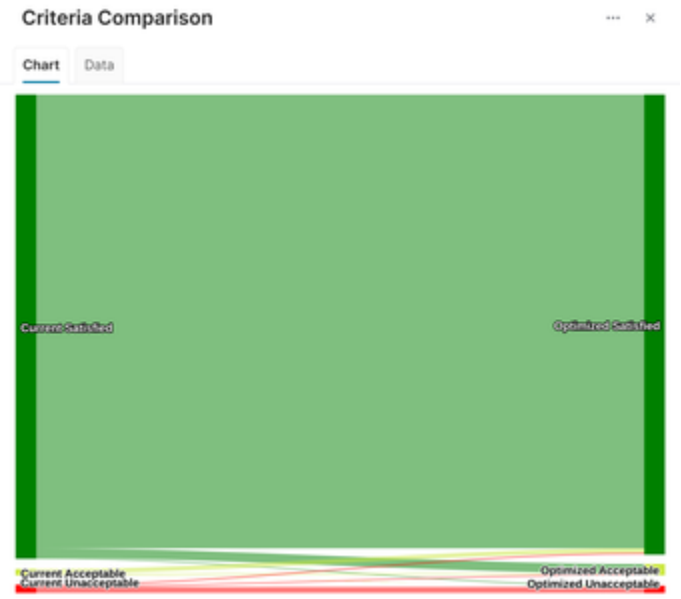
Satisfaction by Criteria Type

Because some criteria may be much more important than others, the criteria satisfaction is also displayed by criteria type. For instance, the global `RevenueProfitMixMaximization` criterion is only a single criterion but it impacts the whole scope of the optimization, whereas there is the `GrossMarginLowerThreshold` criterion for each product x store x period set 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.



Criteria Comparison

Because some criteria could be satisfied in the current state end less acceptable after the optimization, to compensate for other criteria that go from an unacceptable state to an acceptable or satisfied one, the tab provides a Sankey chart. This chart shows the global journey of the satisfaction state of all the criteria.



Evaluation

This tab simulates the evaluation logic that can be called from any other module. One row represents one visible element of the logic. The query that can be done from any other module of the partition is:

```
api.model("myModelName").evaluate(
  "query_results",
  [
    product: "myProductId",
    period_start_date: "myDate",
    store: "myStore"
  ]
)
```

Any of the second parameter keys are optional. The outputs depend on the provided keys. See the details.

Admin User Reference (Optimization - Shelf Price)

Installation (Optimization - Shelf Price)

Optimization - Shelf Price Accelerator deploys a standard Shelf Price Model Class that optimizes the prices in a B2C context.

In this section, you will find all information to deploy and run a standard shelf price optimization. More details on data mapping, elasticity model creation, and optimization results evaluation are in Business User Reference (Optimization - Shelf Price).

Prerequisites

Before you start the installation of the Accelerator, ensure you have **Sales**, **Stock** and **Competition** Datamar ts or Data Sources (as listed in Data Requirements (Optimization - Shelf Price)).

To use the Shelf Price model class deployed by the Accelerator, you will need:

- **Multifactor Elasticity Model** – To create and run a Shelf Price model, you need to generate a Multifactor Elasticity Model which will provide the price elasticities. If not yet deployed on the partition, the Optimization - Multifactor Elasticity Accelerator will be automatically deployed as a dependency of Optimization - Shelf Price Accelerator. To run a Multifactor Elasticity Model, see Prepare Elasticity Model (Optimization - Shelf Price).
- **Optimization Engine (Image)** – To run a Shelf Price model, your partition needs to be allowed to use the Optimization Engine. It will be automatically enabled during the deployment of the Accelerator. In case it is not, please contact Pricefx Support (create a helpdesk ticket) and ask them to enable the image `cr egistry.pricefx.eu/engineering/pricefx-optimization-engine` for your partition, if it is not 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 an increase if it is not enough.

Deployment

1. Access PlatformManager at https://platform.pricefx.com/ and log in with your account or using O365.
2. Go to **Marketplace** > **Accelerator**.
3. Find **Optimization - Shelf Price** Accelerator.
4. Click **Detail**.
5. Select your **Target Partition** from the drop-down menu.
6. Select the required **Accelerator version** you want to deploy and click **Deploy**. Generally, the best version is the last valid one, except when the partition runs too old version of Pricefx.
7. Click **Continue** and wait until the deployment is complete.

Architecture Components (Optimization - Shelf Price)

The Optimization - Shelf Price Accelerator has its own components and also depends on another accelerator. Make sure you review these components and dependencies before the installation.

> This page contains only a components list. For detailed description of the components, see Technical User Reference (Optimization - Shelf Price).

Dependencies

This accelerator depends on Accelerate Multifactor Elasticity Optimization.

Optimization - Shelf Price Accelerator Components

The Optimization - Shelf Price Accelerator consists of the following components which will be deployed to your partition:

Model Class

- Shelf_Price

Calculation Logic

- SP_Def_Eval_Competition
- SP_Def_Eval_Competition_Configurator
- SP_Def_Eval_Sales
- SP_Def_Eval_Sales_Configurator
- SP_Def_Eval_Stock
- SP_Def_Eval_Stock_Configurator
- SP_Sco_Calc_Dataprep
- SP_Sco_Eval_Scope
- SP_Sco_Eval_Scope_Configurator
- SP_Conf_Calc_Aggregation
- SP_Conf_Eval_Boundaries_Configurator
- SP_Conf_Eval_Channels_Configurator
- SP_Conf_Eval_Competition_Configurator
- SP_Conf_Eval_General_Configurator
- SP_Conf_Eval_StockCoverage_Configurator
- SP_Res_Calc_Run_Optimization
- SP_Res_Calc_PrepareResults
- SP_Res_Eval_Details

- SP_Res_Eval_Evaluation
- SP_Res_Eval_Filter_Configurator
- SP_Res_Eval_Glassbox
- SP_Res_Eval_Impact

Groovy Library Logics

- SP_Lib

Technical User Reference (Optimization - Shelf Price)

This section details the Model Class and the logics that the Optimization - Shelf Price 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 Optimization Accelerator Customization and to documentation in Problem Modeling, Problem Description, and Problem Tables.

In this section:

Shelf Price Model Class

The Shelf Price Model Class organizes a list of logics to create the model architecture. It is transformed into an UI in the Pricefx platform that is organized in 4 steps:

1. **Definition** – Maps the sources of data and filters out invalid values.
2. **Scope** – Sets the scope of the optimization.
3. **Configuration** – Sets the parameters of the optimization.
4. **Results** – Looks at the outputs of the optimization.

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 Optimization - Shelf Price Accelerator follow a standard naming convention: first *SP_* prefix, then the first letters of the step name, then *Calc* or *Eval*, depending on the formula nature, then the name of the tab. In the end, there is a library logic named *SP_Lib*.

Library

The logic is **SP_Lib**.

Aim of the logic

This logic 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,

preprocessing the data for the charts, and many small helpers for the charts rendering. The elements `ParametersUtils`, `LabelsUtils`, and `TablesUtils` contain the names of many elements and fields of the models.

- Common reasons to modify the logic
  This lib is the place where to change the names inside the model, to reflect the user business vocabulary. Here you can also write a function to be used in different places of the model class.

Definition Step

There is no calculation logic run in this step. The tabs are **Sales, Stock**, and **Competition**, and their related logics are `SP_Def_Eval_Sales`, `SP_Def_Eval_Sales_Configurator`, `SP_Def_Eval_Stock`, `SP_Def_Eval_Stock_Configurator`, `SP_Def_Eval_Competition` and `SP_Def_Eval_Competition_Configurator`.

Sales Tab

The logics are **SP_Def_Eval_Sales** and **SP_Def_Eval_Sales_Configurator**.

- Aim of the logics
  These logics define the Data Source and the mapping of the entries for the transactions, in the configurator. The main logic calls the configurator and the code for the dashboard portlets.
- Outputs of the evaluation
  Two portlets show the data that will be materialized in the model (table `sales`) and the filtered-out rows.
- Common reasons to modify the logics
  The mapping could be changed if a field is removed or added.

Stock Tab

The logics are **SP_Def_Eval_Stock** and **SP_Def_Eval_Stock_Configurator**.

- Aim of the logics
  These logics define the Data Source and the mapping of the entries for the stock data, in the configurator. The main logic calls the configurator and the code for the dashboard portlets.
- Outputs of the evaluation
  Two portlets show the data that will be materialized in the model (table `stock`) and the filtered-out rows.
- Common reasons to modify the logics
  The mapping could be changed if a field is removed or added.

Competition Tab

The logics are **SP_Def_Eval_Competition** and **SP_Def_Eval_Competition_Configurator**.

- Aim of the logics
  These logics define the Data Source and the mapping of the entries for the competition data, in the configurator. The main logic calls the configurator and the code for the dashboard portlets.
- Outputs of the evaluation
  Two portlets show the data that will be materialized in the model (table `competition`) and the filtered-out rows.

Common reasons to modify the logic

The mapping could be changed if a field is removed or added.

Scope Step

The calculation logic is **SP_Sco_Calc_Dataprep** and there is one tab called **Scope**.

Calculation: Data Preparation

The logic is **SP_Sco_Calc_Dataprep**.

Aim of the logic

This logic validates some prerequisites and materializes the Data Sources in the three tables `sales`, `stock`, and `competition`.

The prerequisites are the consistency of the types of the product and store fields in the different sources and no negative values for the quantities and the prices.

There is no aggregation in the stored data, except to provide summary values, like revenue per product or per store.

Outputs of the calculation

The tables `sales`, `stock`, and `competition` are created in the model and can be used in further steps. These tables are 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, such as mandatory filters. If the waterfall structure is not the standard one, maybe some columns should be added to the `sales` table. In this case, you may also change the user inputs to define this mapping, too (see https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/4425646115#Definition-Step).

Scope Tab

The logics are **SP_Sco_Eval_Scope** and **SP_Sco_Eval_Scope_Configurator**.

Aim of the logics

These logics let the end user choose the filters to scope the optimization, through the configurator *SP_Sco_Eval_Scope_Configurator*. It means, for example, creating a model with a scope filtered on a set of product groups or a minimum store revenue. On the right, it displays some charts to evaluate how the scope is defined.

The scope applies to the Sales data only. The Competition and the Stock data are used as far as they join to the scope of the sales.

Outputs of the evaluation

- A map of filters that are called later in the code by `libs.SP_Lib.ScopeUtils.inputs(model)`. The source data query, filtered on the scope, is given by this sample code, in the next parts of the code:

```
def scope = libs.SP_Lib.ScopeUtils
scope.scopedSalesQuery(model, scope.inputs(model))
```

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

Common reasons to modify the logics

The main reason to modify these logics is to enrich the scope outputs with the data from the stock or the competition sources or with different filters. The filtering options that we expose to the user are modified here.

Configuration Step

There is no calculation in this step. It is separated from the previous step only for better user experience. There are five tabs: **General, Price**, **Competition**, **Stock**, and **Channels**.

The logics are **SP_Conf_Eval_General_Configurator**, **SP_Conf_Eval_Boundaries_Configurator**, **SP_Conf_Eval_Competition_Configurator**, **SP_Conf_Eval_StockCoverage_Configurator**, **SP_Conf_Eval_Channels_Configurator**.

Aim of the logic

The Configuration 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 five tabs is mainly for user experience purposes. Each tab has a different meaning.

Tab	Logic	Aim
General	SP_Conf_Eval_General_Configurator	Sets the most global optimization inputs.
Price	SP_Conf_Eval_Boundaries_Configurator	Sets the goals for the shelf prices: change limits and fixed limits.
Competition	SP_Conf_Eval_Competition_Configurator	Sets the targets of the gap between the model shelf prices and the competition ones, at a product x store level and in average.
Stock	SP_Conf_Eval_StockCoverage_Configurator	Sets the targets of the stock coverage, by product pareto value.
Channels	SP_Conf_Eval_Channels_Configurator	Sets the targets of the average gap by product between a reference channel and the other ones

Outputs of the evaluation

The user inputs are stored to be aggregated in the following step with the rest of the data. The problem tables of the Optimization Engine use the configuration input to define the optimization goals.

Common reasons to modify the logic

The smallest common change is the change of the default values.

It is also 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 tabs in the configuration step, but then the Model Class definition has to be modified too.

Results Step

The calculation logics are **SP_Res_Calc_Run_Optimization** and **SP_Res_Cal_PrepareResults**. There are four tabs: **Impact**, **Details**, **Glassbox** and **Evaluation**.

Calculation: Run Optimization

The logic is **SP_Res_Calc_Run_Optimization**.

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:

- **Validation** of the elasticity model.
- **Data manipulation** to prepare the last tables needed by the OE. These logics are prefixed by *Create_* 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 library function `problemTable` creates automatically such a well-named problem table. 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". That is why it is normal to refactor and improve the create table elements 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. The specific Shelf Price Optimization problem is described as a chart in the logics repository of the accelerator.
- **GlassboxConfig.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 used 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 "Create_" and create the model tables that act as an endpoint for the OE. Be careful, *their names follow a strict format*.

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 two OE jobs, simulation and optimization, and is *not related to a Groovy logic*.

- 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.

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*.

Calculation: Prepare Results

The logic is **SP_Res_Calc_PrepareResults**.

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.

- `Create_Glassbox_` logics calculate aggregated metrics on the optimization agents criteria and value finders.
- Various other tables are created to make the calculation of the dashboards of the Results step faster.

Outputs of the calculation

This calculation writes a collection of tables:

- *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.
- *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.
- *stock_coverage* table - Calculates the stock coverage of each product Pareto category, for each period of time.
- *historical* table - Represents the state for all periods of time before the optimization.

- *forecasted* table - Has the same structure as the *historical* table, but provides the optimized values for all future periods of time.
- *details_product_store_period* table - Contains all the information at product x store x period level for all periods of time, including more extended and unit values than there is in *historical* and *forecasted* tables. It is used as the main source of data for most of the Results step charts and tables.
- *impact_competition_positioning* table - Gives the comparisons metrics between the competition prices and the optimized prices.

Common reasons to modify the logic

If the problem description has been changed, the Create_Forecasted element, which refers to many of the tables created by the OE, may change too.

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 **SP_Res_Eval_Impact**.

Aim of the logic

This tab exposes the results of the OE execution in an HTML summary and some complex graphs, to analyze the forecasted values. The data are filtered according to the user entries set by **SP_Res_Eval_Filter_Configurator**. This filter configurator is shared by both Impact and Details tabs.

Outputs of the evaluation

The details of the provided charts are in https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/4426039328/Result+Description+SPG#Impact.

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 **SP_Res_Eval_Details**.

Aim of the logic

This tab clearly shows the output data of the optimization. This way, the user can see the impact of the optimization on every adjusted value. The data are filtered according to the user entries set by **SP_Res_Eval_Filter_Configurator**. This filter configurator is shared by both Impact and Details tabs.

Outputs of the evaluation

Some data tables. For details see https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/4426039328/Result+Description+SPG#Details.

Common reasons to modify the logic

Add, modify, or remove table outputs. In practice, if the problem description changes, this tab should provide tables to reflect the changes in the variables and constraints. The tables displayed here should allow the end user to access any useful information related to the optimization.

Glassbox Tab

The logic is **SP_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 finetune 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/ACCDEV/pages/4426039328/Result+Description+SPG#Glassbox.

- Common reasons to modify the logic

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

Evaluation Tab

The logic is **SP_Res_Eval_Evaluation**.

- Aim of the logic

  This tab mocks the model evaluation.

  The evaluation 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(“TheModelUniqueName”)
def results = model.evaluate(
  “query_results”,
  [
    product: "someProductID",
    period_start_date: "aDate",
    store: "someStoreID",
  ]
)["Data"]
```

  The product ID, Period Start Date, and Store ID are all optimal keys. The output of the evaluation is a result matrix with columns `product`, `period_start_date`, `store`, and `unit_shelf_price`, filtered on the values provided by the input parameters.

- Outputs of the evaluation

  The output of the evaluation is a result matrix with columns `product`, `period_start_date`, `store`, and `unit_shelf_price`, filtered on the values provided by the input parameters.

- 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.

  It is also possible to add other evaluators to the same model.